AN ABSTRACT OF THE THESIS OF

JEFFREY DAVID BALLANCE for the MASTER OF SCIENCE
(Name) (Degree)

in MATHEMATICS presented on May 23, 1972
(Major) (Date)

Title: MINITRAN: AN ON-LINE FUNCTION TRANSLATOR WITH CAPABILITIES FOR SOLVING ORDINARY DIFFERENTIAL EQUATIONS

Redacted for Privacy

Abstract approved: _____
Joel Davis

A function translator is presented which was designed for interactive programs which allow functions to be defined on-line. The translator handles functions which are specified by a formula and functions which are specified as the solution to a system of differential equations.

MINITRAN: An On-Line Function Translator with Capabilities for Solving Ordinary Differential Equations

by

Jeffrey David Ballance

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1973

APPROVED:

Redacted for Privacy

Associate Professor of Mathematics

in charge of major

Redacted for Privacy

Acting Chairman of Department of Mathematics

Redacted for Privacy

Dean of Graduate School

Date thesis is presented May 23, 1972

Typed by Clover Redfern for Jeffrey David Ballance

ACKNOWLEDGMENTS

I wish to extend my sincere appreciation to Dr. Joel Davis for his advice and helpful suggestions during the preparation of this thesis.

I also wish to thank Mrs. JoAnn Baughman, and Mr. Lyle Ochs of the computer center for their encouragement throughout my graduate program.

I wish to acknowledge the support of the National Science Foundation grant, (GJ 28453), which aided in the development of this system.

TABLE OF CONTENTS

	Page
INTRODUCTION	1
DESCRIPTION OF THE LANGUAGE	5
COMPILATION OF FUNCTIONS	16
Parsing a Function	16
Code Generation	23
Handling of Differential Equations	28
NUMERICAL METHODS	36
Starting Method	36
Predictor-Corrector Methods	41
Numerical Differentiation	58
Numerical Integration	59
BIBLIOGRAPHY	61
APPENDICES	
Appendix A: Modified Backus Normal Form Specification of the Language	63
Appendix B: Calling Sequences for System Routines	66
Appendix C: Flow Charts	71

LIST OF FIGURES

Figure		Page
1.	A system of differential equations entered for the NAPSS system.	3
2.	The same system of equations as in Figure 1 being entered for MINITRAN.	3
3.	The same system of equations entered for Gear's system.	4
4.	This figure shows the format of the pseudo-code table entries.	18
5.	A simplified flowchart of the parsing phase showing the use of the left and right precedences.	21
6.	Pseudo-code table for the expression A*B + T↑2$.	22
7.	A flowchart of the algorithm for suppressing unnecessary temporary storage locations.	26
8.	The pseudo-code table after parsing the first segment of Y(T) = B*C IF T # 0 ELSE 0 $.	27
9.	The format of words in the initial condition array for numerical responses and expression responses.	33
10.	A diagram of the function table entries and pointers for the equations defined in the example.	35
11.	The stability region for the fourth order Adams corrector formula.	44
12.	The stability region for the modified predictor-corrector method.	49
13.	The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.813$.	50
14.	The error bound for the calculated solution shown in Figure 13.	50

Figure		Page
15.	The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.806$.	51
16.	The error bound for the calculated solution shown in Figure 15.	51
17.	The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.794$.	52
18.	The error bound for the calculated solution shown in Figure 17.	52
19.	The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.787$.	53
20.	The error bound for the calculated solution shown in Figure 19.	53
21.	The stability region for the fifth order Adams predictor-corrector method with the corrector being applied only once.	54
22.	Quantities for the largest eigenvalue and the roundoff error are requested with the initial conditions.	57

LIST OF TABLES

Table		Page
1.	A list of the standard functions available to the user in defining functions.	8
2.	Arithmetic and relational operators recognized by MINITRAN.	9
3.	A list of the special symbols and words recognized as separators by MINITRAN.	9
4.	The operators with their external and internal representations and left and right precedences.	19
5.	The operators and their replacements used when optimizing the code.	24

MINITRAN: AN ON-LINE FUNCTION TRANSLATOR WITH CAPABILITIES FOR SOLVING ORDINARY DIFFERENTIAL EQUATIONS

INTRODUCTION

With the growth of time-sharing systems has grown an interest in interactive systems which make it possible for an individual to pose problems to the computer in a "natural" form. On-line, interactive systems provide easy problem investigation without the necessity of learning and using a programming language like FORTRAN. Lyle Smith has written:

> It is, I believe, an accepted fact that machine interaction--both with graphical capability and with only textual output--is a useful, nay necessary, part of our scientific solving repertoire of today and tomorrow (20, p. 262).

Several systems have been developed to make it easier to solve numerical problems on a digital computer without writing a special program. However, very few are able to solve systems of differential equations. The reasons are quite evident and logical. Among the systems that solve differential equations numerically are:

1. MAP (Mathematical Analysis Program) developed at Massachusetts Institute of Technology (14, p. 465-477).
2. NAPSS (Numerical Analysis Problem Solving System) developed at Purdue University (18, p. 51-56).

3. Gear's system for numerically solving differential equations (6, p. 43-49).

The MAP system represents functions by vectors of discrete points and uses parabolic interpolation to get intermediate points. This is not convenient or accurate enough for the solution of large systems of differential equations. The NAPSS system uses poly-algorithms to solve problems, i.e., an algorithm is chosen to give the desired accuracy for the problem. The language is very close to ALGOL with special procedures and statements to do the more sophisticated operations. Gear's system is specifically for the on-line solving of differential equations. It has a natural language for the definitions, but is somewhat restrictive in naming and using the solutions.

MINITRAN is a function translator which was designed to be used as a subsystem of a larger on-line system needing the computational ability. The language in which functions are specified to MINITRAN has features in common with that used by the systems described above and is very close to that used in mathematics. The numerical technique used to solve differential equations is not as versatile as that used by NAPSS or by Gear's system, but is more than adequate for the problems generally encountered.

Figures 1 through 3 illustrate and compare the language used by NAPSS, MINITRAN, and Gear's system to enter the following system

of differential equations:

$$\frac{d^2u}{dt^2} = v + 2\frac{df}{dt} + \sin(g(t) \cdot t)$$

$$\frac{d^2v}{dt^2} = -u + 3.1\frac{dg}{dt} - \cos(f(t))$$

where f and g are previously-defined functions.

```
SOLVE DIFF EQ FOR U(T), V(T)
d↑2U(T) = V(T) + 2dF(T) + SIN(G(T)*T)
d↑2V(T) = -U(T) + 3.1dG(T) - COS(F(T)*T)
T = (0,10), U(0) = V(0) = dU(0) = dV(0) = 0
END
```

Figure 1. A system of differential equations entered for the NAPSS system. Note the implicit multiplication and the d to denote differentiation. The functions F and G were previously defined (18, p. 51-56).

```
U''(T) = V + 2*F'(T) + SIN(G(T)*T)&
V''(T) = -U + 3.1*G'(T) - COS(F(T)*T)$

U = 0
U' = 0
V = 0
V' = 0
```

Figure 2. The same system of equations as in Figure 1 being entered for MINITRAN. Note the numerical differentiation indicated by the primes on the predefined functions F and G. After compiling the equations, MINITRAN requests values for the initial conditions of the problem. The quantities printed by MINITRAN are underlined.

```
BEGIN
PLEASE TYPE EQUATIONS
10 E = (expression for df/dt),
20 F = (expression for f(t)),
30 G = (expression for g(t)),
40 H = (expression for dg/dt),
50 Y0'' = Y1 + 2*E + SIN(G*X),
60 Y1'' = -Y0 + 3.1*H - COS(F*X)
70 END
INITIAL VALUES PLEASE. X =
0.0
        Y0 =
0.0
        Y0' =
0.0
        Y1 =
0.0
        Y1' =
0.0
```

Figure 3. The same system of equations entered for Gear's system. Solutions to differential equations must be Y0 through Y9, only single-letter parameters (not Y or X) are allowed, and the independent variable must be X. Computer responses are underlined (6, p. 43-49).

The following sections discuss the features of the language in which functions are entered, the techniques used to compile the function definitions, and the numerical techniques used by the MINITRAN system. Appendix B contains a description of the calling sequences for the routines a program using MINITRAN would use. The MINITRAN system has been used successfully for over six months in an interactive graphics system used by the Physics Department at Oregon State University.

DESCRIPTION OF THE LANGUAGE

MINITRAN was written in the assembly language COMPASS for the Control Data Corporation 3300 computer as a FORTRAN compatible subprogram. Functions to be translated are passed to MINITRAN as a character string with a dollar sign ($) as the last character in the string. Systems of differential equations are passed as a character string with individual members of the equation set being separated by and signs (&) and with a dollar sign following the last equation in the set.

As output, MINITRAN produces the tables, maps, and machine code necessary to evaluate the functions that have been processed. Although most of the communication with MINITRAN is done by the calling program through the character string containing the function definitions, MINITRAN communicates directly with the user when undefined symbols and initial conditions are to be specified.

MINITRAN allows a user to specify a function in three ways:

a) by a formula, e.g.,

$$F(T) = A*T\uparrow 2 + B*T + C$$

b) as the solution of a single differential equation, e.g.,

$$Y'(T) = -Y$$

c) as the solution to a system of differential equations.

With the exceptions listed below, functions are entered in a notation similar to that of ordinary mathematics. The exceptions are:

a) Implicit multiplication is not allowed. Multiplication is indicated by a single asterisk. Thus, the product $a \cdot b$ is entered as A*B.

b) Subscripting and/or superscripting are not allowed. Exponentiation is signified by the uparrow (↑) or by two asterisks, e.g., a^2 is entered as A↑2 or A**2.

c) No distinction is made between upper- and lower-case letters.

d) The highest derivative term in a differential equation must be isolated on the left of the equal sign.

A function definition can be broken into two parts, with the equal sign as the divider. The quantity to the left of the equal sign is the function identification and gives the name of the function being defined, the order of the derivative if it is a differential equation, and the independent variables. The quantity on the right of the equal sign is a mathematical expression defining the function. An expression is composed of identifiers, constants, operators, and special symbols combined according to the rules of mathematics subject to the restrictions listed above.

An identifier may be the name of a function, parameter, or independent variable. A function name consists of an alphabetic character followed by from zero to three alphanumeric characters.

A parameter or independent variable name may be an alphabetic character followed by zero or one alphanumeric character.

A function may be either a standard function, e.g., SIN, COS, etc. (see Table 1 for a complete list) or be one of the functions the user has already entered. In the case of a system of differential equations, references may be made to any function in the system, as well as to those previously defined.

The form of a constant is the same as that of a constant in a FORTRAN source program. A constant can be entered in scientific notation with the exponent being specified with an E followed by a signed or unsigned integer. For a formal definition of the form of constants, see Appendix A. Some examples of constants are:

1

0.2

2E-3 is equivalent to $2 \cdot 10^{-3}$

100.59

Internally, all constants are represented as floating point numbers. Hence, there is no distinction between 1 and 1.0 .

Arithmetic operators and special symbols are used to combine identifiers and constants in expressions. In addition to the arithmetic operators shown in Table 2, the differential operators ' and '' have been introduced to identify the differential equation solution to be used or to specify that the numerical derivative of the function is to

Table 1. A list of the standard functions available to the user in defining functions.

SIN(X)	sine function
COS(X)	cosine function
SQRT(X)	square root function
ABS(X)	absolute value function
COSH(X)	hyperbolic cosine function
SINH(X)	hyperbolic sine function
TANH(X)	hyperbolic tangent function
TAN(X)	tangent function
ASIN(X)	inverse sine function
ACOS(X)	inverse cosine function
ATAN(X)	inverse tangent function
EXP(X)	exponential function--e^x
LN(X)	base e logarithm
LOG(X)	base 10 logarithm
INT(F, A, B, N)	calculates $\int_a^b f(t)dt$ using a quadrature formula with n intervals
SUM(F, K, A, B, I)	computes $\sum_{k=a}^{b} f$ with k being incremented by i
ERR(Y)	returns the computed upper bound for the total error in the system of differential equations involving Y

be used. The special symbols (,), [, and] are used to group operations in the normal way. Several symbols and words are used as separators. These are listed with their descriptions in Table 3.

Table 2. Arithmetic and relational operators recognized by MINITRAN.

** or ↑	exponentiation
*	multiplication
/	division
+	addition and unary plus
-	subtraction and unary minus
<	less than relation
>	greater than relation
=	equal relation
<=	less than or equal relation
>=	greater than or equal relation
#	not equal relation

Table 3. A list of the special symbols and words recognized as separators by MINITRAN.

$	signals the end of a string to be processed by MINITRAN
&	used as a separator between individual members of a system of equations
; or ELSE	separates segments of a function defined by conditionals
IF	separates an expression giving a possible value for the function from the conditions under which the function will be defined by that expression

Before discussing more specific features of the language, a few examples will be given to illustrate how functions specified by a

formula are entered for MINITRAN. In each of the examples, note that a parameter does not have to be defined before it is used. After compilation, parameters which have not appeared in previously-entered functions will be printed by MINITRAN and their values requested. In the following example, the underlined quantities are printed by MINITRAN.

```
Y(T) = -0.5*G*T↑2$

       G = 9.8

F(T) = EXP(SIN(T)*Y(T)) - 2*COS(T/2)$

H(X, Y) = -G*X↑2 + B*F(Y)$

       B = 5.27
```

Two things should be noted about the function H(X, Y) in the example. First, a function may have more than one independent variable. Second, if an independent variable has the same name as a function already defined, no ambiguity results since independent variables have been given higher priority than functions.

Many times a function is defined by several different formulas, depending upon certain conditions. To handle these functions a conditional statement has been provided, which has a form very close to that used in textbooks. For example, the function

$$f(t) = \begin{cases} \sin(t) & 0 \leq t \leq \pi/2 \\ 1 & \pi/2 < t < \pi \\ e^{t-\pi} & \pi \leq t \end{cases}$$

would be entered for MINITRAN in the following way (the word "ELSE" can be used instead of the semicolon):

```
F(T) = SIN(T) IF 0 <= T <= PI/2;
       1 IF PI/2 < T < PI;
       EXP(T-PI) IF PI <= T$
```

The parameter PI is supplied by the program and has the value 3.14159... . If, in a function using conditionals none of the conditions are satisfied, a zero value is returned.

One of the special features allowed by MINITRAN is the option of omitting the argument list of a function appearing in an expression if it is the same as that of the function being defined. For example, if one wished to use the function Y(T) from the example in the function $u(t) = y(t) \cdot t^2$, one could enter

```
U(T) = Y*T↑2$
```

instead of

```
U(T) = Y(T)*T↑2$
```

This feature cannot be used if the functions have a different number of independent variables.

The normal algebraic interpretation was maintained for expressions using the unary plus and minus. For example, the expression -A↑-B is interpreted as $-(a^{-b})$. Note in the above example that a unary plus and minus may follow another operator. Other examples of the use of this feature are:

A↑-B means a^{-b}

A*-B means a(-b)

A+-B means a+(-b)

An approximation to the first or second derivative of a function may be used in an expression by following the function name with ', ", or two single primes. For example, if F(T) had been defined as in the example on page 10, entering

```
G(T) = 5*F' + T↑2$
```

will cause the first derivative of F(T) to be calculated when G is evaluated. Only the first or second derivative of a function specified by a formula may be specified in the current version of MINITRAN. It is not possible to calculate higher derivatives by successive function definitions. For example, the following construction is not allowed:

```
F(T) = T↑2$

H(T) = F'(T)$

G(T) = H'(T)$
```

In addition to specifying a function by a formula, a function may be specified as the solution to a differential equation or a system of differential equations. These functions are entered in nearly the same way as functions specified by a formula.

When entering a differential equation, the highest derivative term is isolated on the left of the equal sign. The highest derivative term with the independent variables thus becomes the function identification. For example, to enter the differential equation

$$y' + y^2 - t^2 = 0$$

one would rewrite it as

$$y' = t^2 - y^2$$

and enter it as

```
Y'(T) = T↑2 - Y↑2$
```

A system of differential equations is entered the same as a single equation, except that the members are separated by <u>and</u> signs (&). For example, the system

$$y' - x^2 - y^2 = 0$$

$$x' - x^2 + y^2 = 0$$

would be entered as

```
Y'(T) = X↑2 + Y↑2&
X'(T) = X↑2 - Y↑2$
```

The rules for entering the expression on the right of the equal sign for these functions are the same as those discussed previously for functions specified by a formula. To simplify later discussions, systems of differential equations will be defined to include the case of a single differential equation.

After a system of differential equations has been entered and compiled, the initial conditions for the equations are requested, along with any parameters which may be undefined. For example, the above system would appear as

```
Y'(T) = X↑2 + Y↑2&
                                  (equations passed to MINITRAN)
X'(T) = X↑2 - Y↑2$

  Y = 0

  X = 0.5
```

In response to the requests for initial conditions in the above example, numerical quantities were entered. However, the response may be an expression as well. This feature only applies to requests for initial conditions, and does not apply to requests for parameter values. An example of this feature is:

```
X"(T) = 0&
                        (equations passed to MINITRAN)
Y"(T) = -G$

  X = 0

  X' = VO*COS(TH)$

  Y = 0

  Y' = SQRT(VO↑2 - X'↑2)$

  G = 9.8

  VO = 20

  TH = .34
```

As illustrated by the example, the expression specified in response to a request for an initial condition may involve other initial conditions. These expressions are evaluated in the order they are entered; hence, one cannot have an expression involving an initial condition which will be specified later with an expression. In the above example, Y' could not have appeared in the expression for X'.

COMPILATION OF FUNCTIONS

The translation of an alphanumeric string to executable code has been divided into two phases. During the first phase, the string is parsed to determine if there are any errors and to generate a table of the operands and operators to be used in generating machine code. If an error is detected during the scan, an appropriate message is printed and compilation is terminated. During the second phase, machine code is generated from the table of operators and operands and stored in the memory area.

Parsing a Function

The initial part of the parsing is to determine the name of the function, the names and number of the independent variables, and whether the function is specified by a formula or as the solution to a system of differential equations. If a system of differential equations has been entered, the other function names in the system must be determined. It is assumed that a system of differential equations will be passed to MINITRAN with and signs (&) separating the individual members and with a dollar sign ($) at the end of the last equation as with any function. For example, the system of differential equations

$$X'(T) = X\uparrow 2 + Y\uparrow 2$$

$$Y'(T) = X\uparrow 2 - Y\uparrow 2$$

when passed to MINITRAN would appear as:

X'(T) = X↑2 + Y↑2&Y'(T) = X↑2 - Y↑2$

while the function

F(T) = T↑2

when passed to MINITRAN would appear as:

F(T) = T↑2$

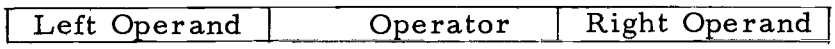
After determining the name of the function and the names of the independent variables, the expression defining the function is parsed. As the expression is parsed, a table of pseudo-code is developed. This table contains three-item entries of the form

<left operand ><operator ><right operand >

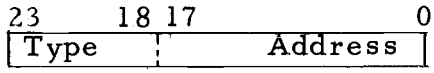
Figure 4 shows the format of these entries.

To insure that the operations are done in the correct order, a left precedence and a right precedence has been assigned to each operator (1, p. 222-239). Giving each operator two precedences allows certain operations to be done from right to left instead of left to right. For example, in the expression A↑B↑C the right precedence of the uparrow operator is greater than that of the left precedence, so that the expression is evaluated as $a^{(b^c)}$ instead of $(a^b)^c$. Table 4 shows the right and left precedences for each operator.

Pseudo-Code Table entry:



Operand word format:



Type:

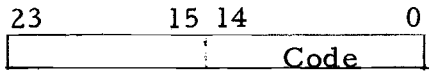
00_8 - temporary storage indicator

01_8 - constant

02_8 - parameter

40_8 - function

Operator word format:



Each of the operators has a code in the range $0\text{-}32_8$

Figure 4. This figure shows the format of the pseudo-code table entries.

To facilitate the use of the precedence rules, two last-on-first-off stacks are maintained; one is for operands and the other is for operators. Each operand is placed on the operand stack when it is encountered. When an operator is encountered, its right precedence is compared with that of the left precedence of the top operator on the stack. The new operator will be stacked if its right precedence is

Table 4. The operators with their external and internal representations and left and right precedences.

Operation	External Representation	Internal Representation	Left Precedence	Right Precedence
IF	IF	:	0	1
ELSE	ELSE	;	0	1
End of equation in system	&	&	0	1
End of string to be processed	$	$	0	1
Assignment	=	←	1	2
Comma	,	,	1	2
Less than	<	<	4	4
Greater than	>	>	4	4
Not equal	#	#	4	4
Greater than or equal	>=	≥	4	4
Less than or equal	<=	≤	4	4
Equal	=	=	4	4
Plus	+	+	5	5
Minus	-	-	5	5
Multiplication	*	*	6	6
Division	/	/	6	6
Exponentiation	** or ↑	↑	7	8
Unary minus	-	÷	7	8
Unary plus	+	(ignored)		
Function		f	8	9

greater than that of the left precedence of the top operator; otherwise, operators and operands will be unstacked until the new operator can be stacked.

When an operator is unstacked, it is placed in the pseudo-code table with the top two operands from the operand stack. It is assumed that the result of each operation will need to be stored in a temporary location as an operand for another operator. Hence, after unstacking an operator and its operands and placing them in the pseudo-code table, a temp indicator is incremented and placed on the operand stack. To insure that the first operator will be handled correctly, the operand stack is initialized with a constant zero as its first entry and the operator stack is initialized with an end-of-line operator and a replacement operator as its first entries.

Figure 5 shows a simplified flowchart of the parsing phase. It is suggested that the reader use the precedences shown in Table 4 and the flowchart to parse the expression

A*B + T↑2$

and to generate the pseudo-code. The resulting pseudo-code table is given in Figure 6.

The parsing of a function involving conditionals differs in some respects from that of functions without conditionals. The general forms of the conditional are:

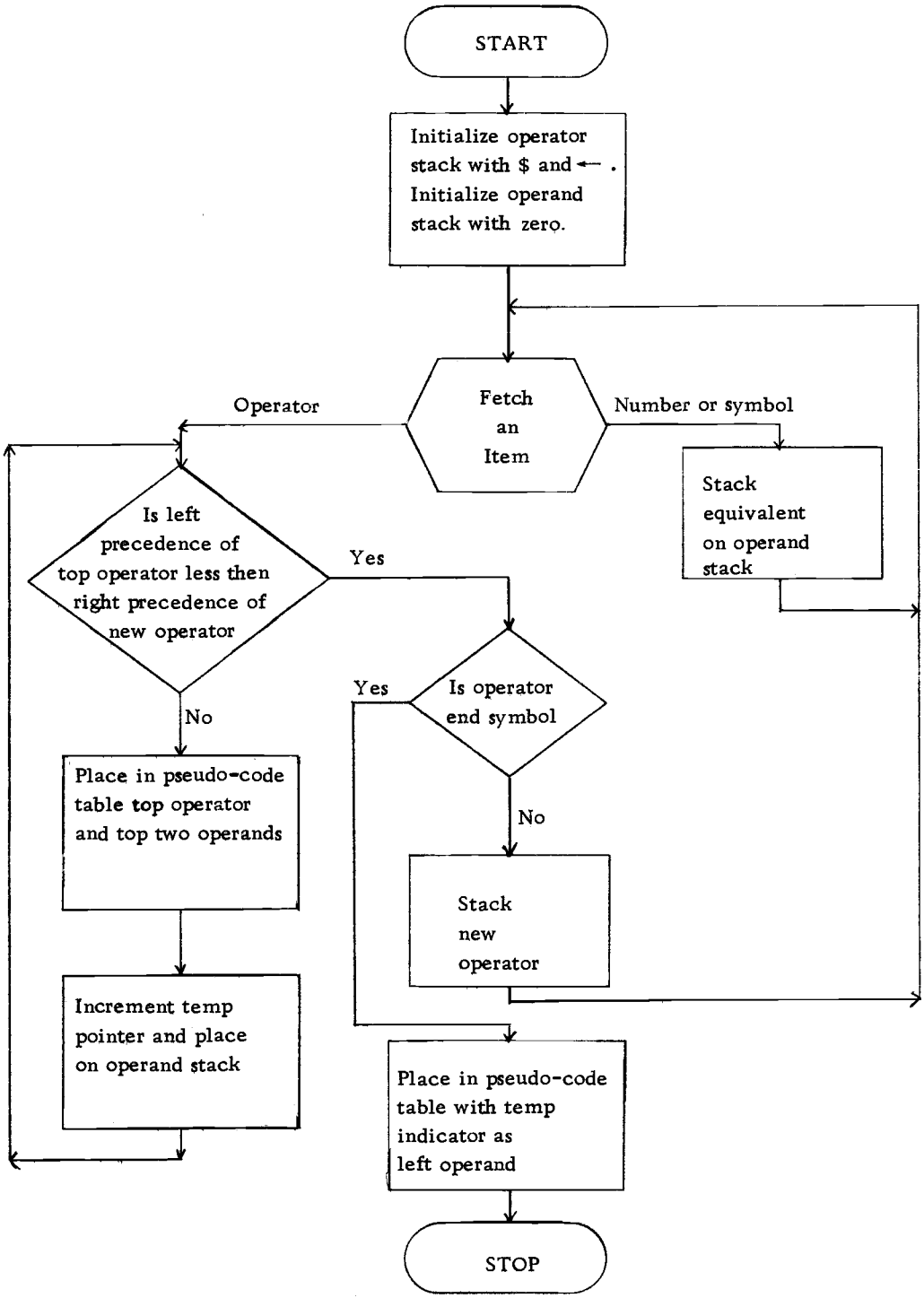


Figure 5. A simplified flowchart of the parsing phase showing the use of the left and right precedences.

a) <expression> IF <expression><relation><expression>

b) <expression> IF <expression><relation><expression>

<relation><expression>

The expression giving a possible value for the function is separated from the conditions under which the function will take on that value by an IF. The IF operator has been given the same precedences as the end symbols &, $, and ; (ELSE) to cause the operator and operand stacks to be emptied. After the stacks have been emptied by an IF or an end symbol, a test is made to see if the IF operator is the one being processed. If it is, the parse is continued until an ELSE or other end symbol is encountered before going on to generate the code. Thus, each segment of a function defined by conditionals is completely processed before continuing to the next, i.e., both phases of compilation are completed on each segment before the parsing of the next segment is begun. This approach was used because these functions tend to be long, and if parsed as a whole would require that the pseudo-code table be unnecessarily large.

A	*	B
T	↑	2
Temp 1	+	Temp 2
Temp 3	$	

Figure 6. Pseudo-code table for the expression A*B + T↑2$.

During the parsing of a conditional, operators and operands are stacked and unstacked in the way described above, with one exception. After a relational operator and its operands are unstacked and placed in the pseudo-code table, the right operand is placed back on the operand stack. This insures that the correct operand will be used by the following relational operator. The reader can verify that the relational expression:

A < B < C$

would produce the following entries in the pseudo-code table:

A	<	B
B	<	C
Temp 0	$	

Code Generation

After an expression has been parsed and the table of pseudo-code created, the code generation phase is entered. In addition to generating machine code from the pseudo-code, some optimization is done.

In optimizing the code, the major emphasis is placed on reducing the number of times an intermediate result must be stored in a temporary location. The algorithm used to do this is described below and is quite straight-forward and easily implemented.

Again, as in the parsing phase, it is assumed that the result of each operation must be stored in a temporary location. A temp indicator is maintained in the same way as in the parsing phase. However, before the store instruction is generated the temp indicator is compared with the operands of the next operator. If the temp indicator and the left operand are the same, the store instruction is not generated and the code for the operator is generated. If the temp indicator is not the same as the left operand, but is the same as the right operand, a test is made to see if the operation is commutative and the operands can be interchanged. Most of the operators are not commutative, but it is possible in many cases to replace the operator with a complimentary operator. For example, the relation A < B can be represented equivalently as B > A. Table 5 lists the operators that can be changed and their replacements.

Table 5. The operators and their replacements used when optimizing the code.

Operator	Replacement
+	+
-	⍥ (reverse minus)
←	←
*	*
⍥ (unary minus)	⍥ (reverse minus)
>	<
<	>
# (not equal)	=
=	# (not equal)
≥	≤
≤	≥

If the temp indicator is equal to the right operand of the operator and the operands can be interchanged, the instruction to store into a temporary location is not generated. The operands are interchanged, the operator replaced, and the code for the new operator is generated. If the temp indicator is not equal to either operand, an instruction must be generated to store the result in a temporary location.

Figure 7 shows a simplified flowchart of the algorithm described above. The reader can verify that if the algorithm were used in generating code for the following expression

A*B↑2 + C$

no temporary storage locations would be necessary.

Probably the most common use of the exponentiation operator is in squaring a quantity, e.g., X↑2 and (A + B)↑2. Since it is much faster to multiply the quantity by itself than it is to call an exponentiation routine, a test is made for an exponent of 2 when the exponentiation operator is handled, and the appropriate code generated.

A special routine has been supplied and is called when exponentiation is to be done. This routine tests for a negative base being raised to an integer power and then calls the appropriate FORTRAN library routine. This special test is necessary, since all quantities are assumed to be real by MINITRAN and the library exponentiation

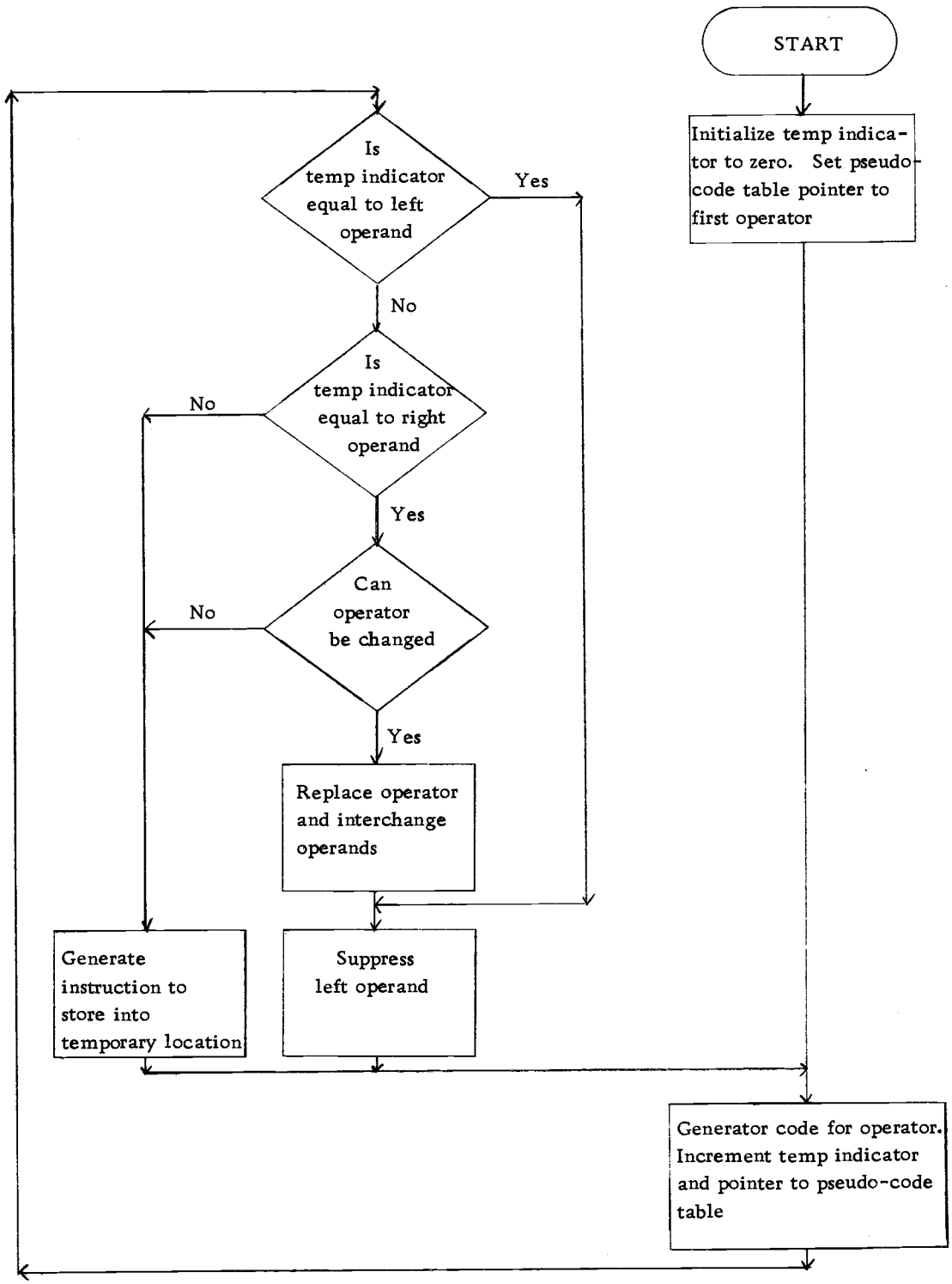


Figure 7. A flowchart of the algorithm for suppressing unnecessary temporary storage locations.

routine uses logarithms to compute the result of raising a real base to a real power.

The order in which operators are taken from the pseudo-code table is different for functions involving conditionals, due to the form in which they are entered. As described previously, the IF operator is used as a separator and has the same precedences as an end symbol. Hence, the entries for the expression giving the value of the function will precede the IF operator in the pseudo-code table and the entries for the conditions will follow the IF operator. Figure 8 shows the pseudo-code table after parsing to the ELSE in:

Y(T) = B*C IF T # 0 ELSE 0 $

In order to generate the code so that the conditions are tested first, the code for the entries after the IF operator is generated and then the code for the entries preceding the IF operator is generated.

B	*	C
Temp 1	:	
T	#	0
Temp 2	;	

Figure 8. The pseudo-code table after parsing the first segment of Y(T) = B*C IF T # 0 ELSE 0 $.

Handling of Differential Equations

The code generation for systems of differential equations differs very little from that of functions specified by a formula. The major difference is that a special routine for the system is inserted after the last function in the equation set has been processed. This routine is called when the system of equations is solved.

As described earlier, once a function has been defined, it may appear in future function definitions. The maps and information necessary to allow this are contained in the function table. The function table has a four-word entry for each defined function, with the first word of each entry containing the name of the function. The information contained in the other three words is different for a function specified by a formula than for a function specified as the solution to a system of differential equations. The information contained in each of the entries is described below.

The entries in the function table for a function specified by a formula are as follows:

Word 2: the address of the routine generated and a code indicating that the function was defined by a formula

Word 3: the number of times this function is used in other function definitions

Word 4: the address of the map of the systems of differential equations referenced by the function

The entry in this word and its use are discussed below. If the function does not reference any differential equation solutions, this word is zero.

The entries in the function table for a function specified as the solution to a system of differential equations are as follows:

Word 2: a pointer to an information block for the system of equations and a code indicating that the function was defined as the solution to a system of differential equations

Word 3: an index giving this equation's position relative to other members in the equation set

This index is given by the sum of the orders and the number of equations which are before it in the equation set. For example, in the differential equation system:

$$X''(T) = X + Y$$

$$Y''(T) = X - Y$$

the index for X would be zero and the index for Y would be three. This index is used to reference the appropriate initial condition or solution of the system of equations.

Word 4: the order of the equation

As indicated above, each system of differential equations has a common block of information pointed to by each member of the equation set. The form in which this information is stored is as follows:

Word 1: the address of the controlling routine for the system of equations

This routine is generated after the last function in the system has been processed and is called when the system is solved.

Word 2: the number of entries in the array containing the initial conditions and a pointer to the array

Word 3: the number of equations in this set of equations and a pointer to an array which will contain the addresses of the solutions when the system of equations is solved

Word 4: contains a pointer to a map containing a pointer to the information block for every system of differential equations referenced

This map always contains a pointer to the information block of the system of equations itself.

Following the fourth word in the information block are words containing the order of each member in the system of equations and the address of the code associated with each.

The differential equation reference map is developed as each function is being translated. Every time a function is referenced, its

differential equation reference map is merged with that of the function being translated. This insures that all systems of differential equations needed in order to evaluate a function are known at the time of evaluation. The map is used when the function is evaluated to combine the individual systems of equations into one large system. The resulting system of equations is then solved simultaneously by the differential equation solving routine.

The solutions to systems of differential equations are stored consecutively in an array with no distinction made between individual systems. The array pointed to by word 3 of the information block is thus initialized to contain the addresses of the solutions to the particular system of equations it represents. For example, if the system

$$X''(T) = X + Y$$

$$Y''(T) = X - Y$$

were one of the systems solved, the entries in this array would contain the addresses of the solutions in the order X, X', X", Y, Y', and Y".

Using indirect addressing, it is very easy to get a particular solution. The address of the word containing the solution address is computed by adding the following quantities:

a) the number of primes following the function name, i.e., if the solution X' of a system were desired, the number of

primes would be one,

b) the system index found in word three of its function table entry,

and c) the address of the array containing the solution addresses.

When a differential equation solution is used in a function definition, the routine is called whose address is in word 1 of the information block for the system of equations. This routine expects the address of the word containing the address of the desired solution to be passed to it. It can thus return the appropriate solution value after calling the routine which solves the system of differential equations for the current value of the independent variable.

As described earlier, the initial conditions for a system of differential equations are requested after successful compilation. The numeric responses are stored in the initial condition array pointed to by word 2 of the information block for the system of equations. When an expression is given for an initial condition, it is parsed and the appropriate code generated. The address of the routine generated is stored in the initial condition array in the format shown in Figure 9, and can be easily recognized when the differential equation solver is being initialized to solve the system of equations.[1] The routine is

[1] This form restricts positive numbers given in response to a request for an initial condition to be larger than $1.5 \cdot 10^{-306}$.

then evaluated and the resulting value used as an initial condition for the system of equations.

Floating point response

Expression response

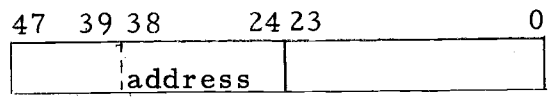


Figure 9. The format of words in the initial condition array for numerical responses and expression responses.

The following example will help to clarify the previous discussion of the function table entries and differential equation reference maps. Two differential equations and an equation referencing both are entered as shown below (underlined quantities are printed by the computer):

Y"(T) = -K*Y - B*Y'$ (equation passed to MINITRAN)

Y = 1.0

Y' = -0.25

K = 0.5

B = 2.5

X"(T) = -K1*X - B1*X'$ (equation passed to MINITRAN)

X = 1.0

X' = 1.0

K1 = 0.75

B1 = 1.75

E(T) = SQRT(X↑2 + Y'↑2)$ (equation passed to MINITRAN)

Figure 10 shows the function table entries, information blocks for the differential equations, and the reference maps with the various pointers indicated. The reference map for E(T) contains entries for both the X and Y differential equations. When E(T) is to be evaluated, the differential equation solver is initialized to solve these systems simultaneously.

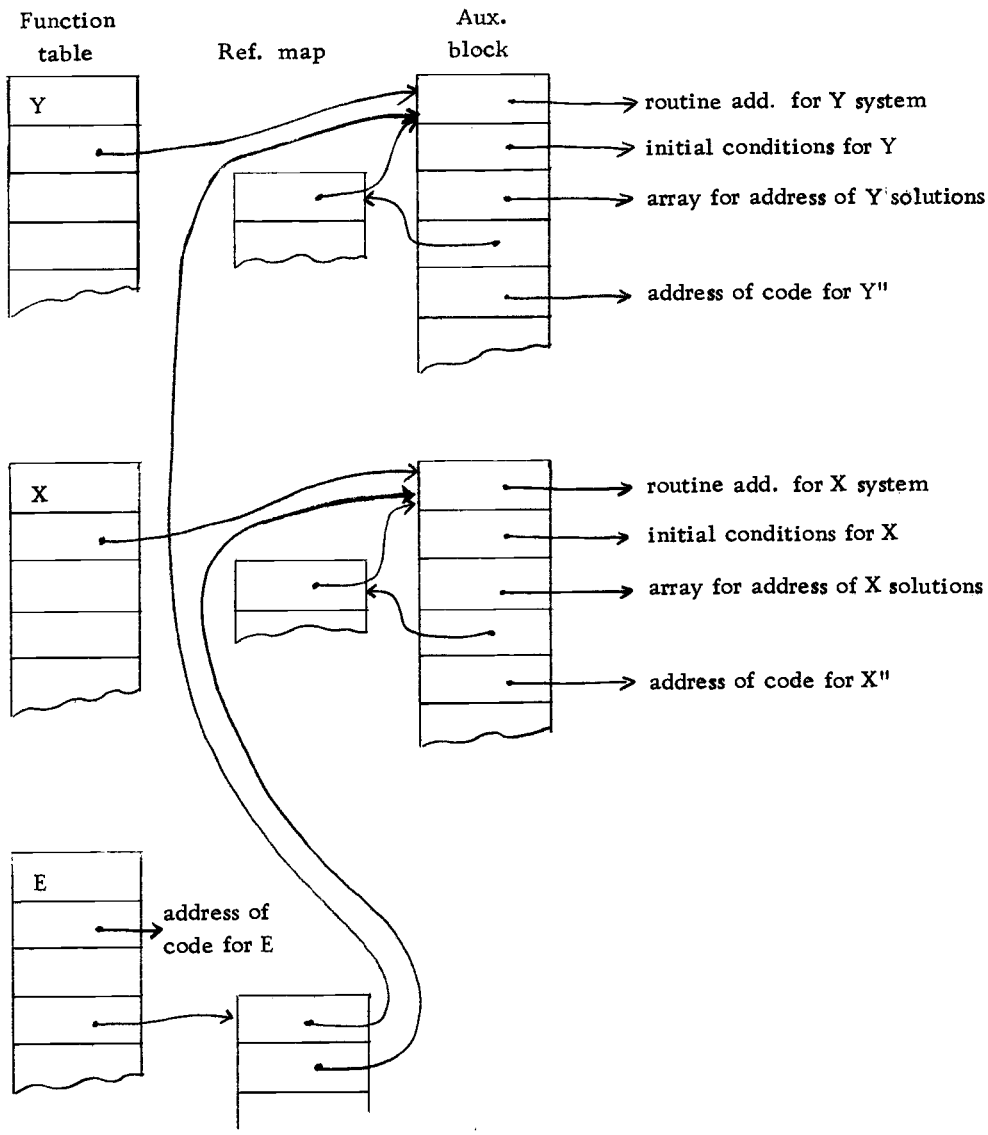


Figure 10. A diagram of the function table entries and pointers for the equations defined in the example.

NUMERICAL METHODS

The numerical technique used to calculate the solutions for systems of differential equations is an Adams predictor-corrector method. Before discussing this method and the way it is used in the program, a discussion is given of the technique adopted for calculating the first few values needed by the predictor-corrector formulas.

Starting Method

The usual procedure used to calculate the first few values needed by a predictor-corrector method is a Runga-Kutta method. However, the Runga-Kutta formulas have very complex truncation error terms. Ralston (17, p. 116) gives the following bound for the local truncation error of a fourth order Runga-Kutta method:

$$T \leq \frac{73}{720} ML^4 h^5$$

where in a region containing (x_0, y_0)

$$|f(x, y)| \leq M$$

$$\left| \frac{\partial^{i+j} f}{\partial x^i \partial y^j} \right| \leq \frac{L^{i+j}}{M^{j-1}}$$

where M and L are positive constants independent of (x, y) and $(i+j) \leq 4$. Thus, the truncation error depends on partial derivatives

which may not behave properly.

The approach adopted in the program was suggested by W.E. Milne (16, p. 42-47) in his book on numerical solutions to differential equations. The general approach is to "guess" values for the points needed and then to refine them using some iterative formulas of the appropriate order.

In the following discussion, y_i and f_i have been defined as:

$$y_i = y(t_0+ih)$$

$$f_i = y'(t_0+ih)$$

where $y(t_0+ih)$ is the solution vector, $y'(t_0+ih)$ is the problem vector, t_0 is the initial value of the independent variable and is assumed to be zero, and h is the stepwidth. The iterative formulas were developed from the general formula:

$$y_j = y_0 + h(a_j f_0 + b_j f_1 + c_j f_2 + d_j f_3) \quad j = 1, 2, 3$$

The coefficients were determined by the method of undetermined coefficients, i.e., the formulas were made exact for the problem

$$y(t) = t^k \qquad k = 0, 1, 2, 3, 4$$

After determining the a's, b's, c's and d's, the truncation error terms were calculated by applying

$$y(t) = t^5$$

The resulting formulas are:

$$y_1 = y_0 + \frac{h}{24}(9f_0+19f_1+5f_2+f_3) - \frac{19h^5 f^{(4)}(\xi)}{720} \qquad (1a)$$

$$y_2 = y_0 + \frac{h}{3}(f_0+4f_1+f_2) - \frac{h^5 f^{(4)}(\xi_2)}{720} \qquad (1b)$$

$$y_3 = y_0 + \frac{3h}{8}(f_0+3f_1+3f_2+f_3) - \frac{h^5 f^{(4)}(\xi_3)}{80} \qquad (1c)$$

The procedure for using these formulas is as follows:

a) "Guess" values for y_1, y_2, and y_3

b) Use these values in (1a) to get a better estimate for y_1; the new y_1 is used with the "guessed" quantities y_2 and y_3 to get a better estimate of y_2, etc.

c) Repeat (b) until

$$\underset{j}{\text{Max}} \left\| \frac{y_j^{\text{old}} - y_j^{\text{new}}}{y_j^{\text{new}}} \right\|_\infty < 10^{-10} \qquad j = 1, 2, 3$$

In this case and future occurrences, appropriate modification is made for division by zero.

The initial "guesses" needed for the iterative formulas are calculated using Euler's method:

$$y_{i+1} = y_i + hf_i \qquad (2)$$

The stepwidth h is adjusted when y_1 is calculated so that the iterative method will converge. This is done by calculating two estimates of y_1, using stepwidths h and $h/2$ with Equation (2). Let $\bar{y}_1$ be the estimate calculated with stepwidth $h/2$ and y_1 be the estimate calculated with stepwidth h. The stepwidth is adjusted by halving until

$$\left\| \frac{y_1 - \bar{y}_1}{y_1} \right\|_\infty \leq 10^{-5}$$

All of the starting values y_1, y_2, and y_3 are calculated using stepwidths h and $h/2$, giving estimates y_i and $\bar{y}_i$ respectively $(i = 1, 2, 3)$. The two estimates are used to get a better value by eliminating the $O(h^4)$ term from:

$$y_i = y_i + O(h^4) \tag{3a}$$

$$y_i \cong \bar{y}_i + \frac{O(h^4)}{16} \qquad i = 1, 2, 3 \tag{3b}$$

to get

$$y_i \cong \bar{y}_i + \frac{\bar{y}_i - y_i}{15} \qquad i = 1, 2, 3 \tag{4}$$

It is interesting to compare the number of function evaluations required by the iterative method described above using Euler's method to "guess" the starting values with the number required using

a Runga-Kutta-Gill method to "guess" the starting values. Using Euler's method, the number of function evaluations is given by:

$$M_1 \cdot 2 + N_1 \cdot 3 + 10$$

where

M_1 -- the number of times the initial stepwidth is halved.

N_1 -- the number of times the iterative formulas are used.

$N_1 \geq 1$.

If the Runga-Kutta-Gill method is used to calculate the initial "guesses" for the iterative scheme as is suggested by Ralston (17, p. 101, 117), the number of function evaluations is:

$$M_2 \cdot 8 + N_2 \cdot 3 + 37$$

where M_2 and N_2 are defined in an analogous way as M_1 and N_1 above. In general, it would be expected that $M_2 \leq M_1$ and that $N_2 \leq N_1$. However, unless the differences are fairly great, using Euler's method in the iterative scheme would require fewer function evaluations. In comparing the methods, it was found that using Euler's method did require fewer function evaluations. For example, with the problem

$$y(t) = \frac{5y}{1+t}, \qquad y(0) = 1$$

the number of function evaluations using Euler's method was 29 and was 83 using the Runga-Kutta-Gill method. The initial stepwidth $h = 0.1$ was halved five times by each method.

Predictor-Corrector Methods

Once the starting values are calculated, the predictor-corrector formulas are used to calculate succeeding points. The program allows the user to choose one of two predictor-corrector schemes:

a) Iterated -- The Adams corrector is iterated until the normalized difference between successive corrected values is less than 10^{-10}.

b) Modified Predictor-Corrector -- This method was suggested by Milne (16, p. 65) and involves adding a correction to the predicted and corrected values to compensate for the truncation error terms.

A discussion of each method is presented below.

The iterated method used by the program is given by:

Predict:

$$p_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}) + \frac{251}{720}h^5 f^{(4)}(\xi) \quad (5)$$

Correct:

$$y_{i+1}^{(m+1)} = y_i + \frac{h}{24}(9f_{i+1}^{(m)} + 19f_i - 5f_{i-1} + f_{i-2}) - \frac{19}{720}h^5 f^{(4)}(\xi) \quad (6)$$

$$m = 0, 1, 2, \ldots$$

where

$$y_{i+1}^{(0)} = p_{i+1} .$$

The corrector is applied until

$$\left\| \frac{y_{i+1}^{(m+1)} - y_{i+1}^{(m)}}{y_{i+1}^{(m+1)}} \right\|_{\infty} \leq 10^{-10}$$

Since the corrector has been iterated to convergence, the effect of the initial predicted value on the final corrected value is negligible. Hence, the stability of the method is determined by the corrector alone (2, p. 457-468).

In the discussion to follow, let z_i be the true solution of the problem and define $\epsilon_i = z_i - y_i$ to be the difference between the true solution and the calculated solution. If after being iterated to convergence, Equation (6) is subtracted from the true solution, the following equation is obtained:

$$\epsilon_{i+1} = \epsilon_i + \frac{h}{24}(9(z'_{i+1} - f_{i+1}) + 19(z'_i - f_i) - 5(z'_{i-1} - f_{i-1}) + (z'_{i-2} - f_{i-2})) + E_{i+1} \tag{7}$$

where E_{i+1} is the truncation error term. After applying the Mean Value Theorem to get

$$z'_i - f_i = \frac{\partial f}{\partial y}(z_i - y_i)$$

and assuming that $\partial f/\partial y$ is constant, Equation (7) can be rewritten as

$$\epsilon_{i+1} = \epsilon_i + \frac{\bar{h}}{24}(9\epsilon_{i+1} + 19\epsilon_i - 5\epsilon_{i-1} + \epsilon_{i-2}) + E_{i+1} \qquad (8)$$

where $\bar{h} = h\frac{\partial f}{\partial y}$. This is a difference equation with a solution of the form $\epsilon_i = E\rho^i$. Substituting this into Equation (8) gives the characteristic equation:

$$\rho^3 - \rho^2 - \frac{\bar{h}}{24}(9\rho^3 + 19\rho^2 - 5\rho + 1) = 0 \qquad (9)$$

A multistep method is said to be absolutely stable for values of $\bar{h}$ for which the roots ρ_j of Equation (9) satisfy (5, p. 128):[2]

$$|\rho_j| \leq 1 \qquad j = 1, 2, 3$$

The region of absolute stability for the method can be calculated by rewriting Equation (9) as:

$$\bar{h} = \frac{\rho^3 - \rho^2}{\frac{1}{24}(9\rho^3 + 19\rho^2 - 5\rho + 1)}$$

and solving it for $\rho = e^{i\theta}$, $0 \leq \theta < 2\pi$. The region is shown in Figure 11 plotted in the complex $\bar{h}$ plane.

The analysis for the modified predictor-corrector method of calculating the solution is more difficult than for the iterated method. The method was suggested by Milne (16, p. 65) and involves adding

[2]There must not be any multiple roots of magnitude 1.

a correction to the predicted and corrected values to compensate for the truncation errors.

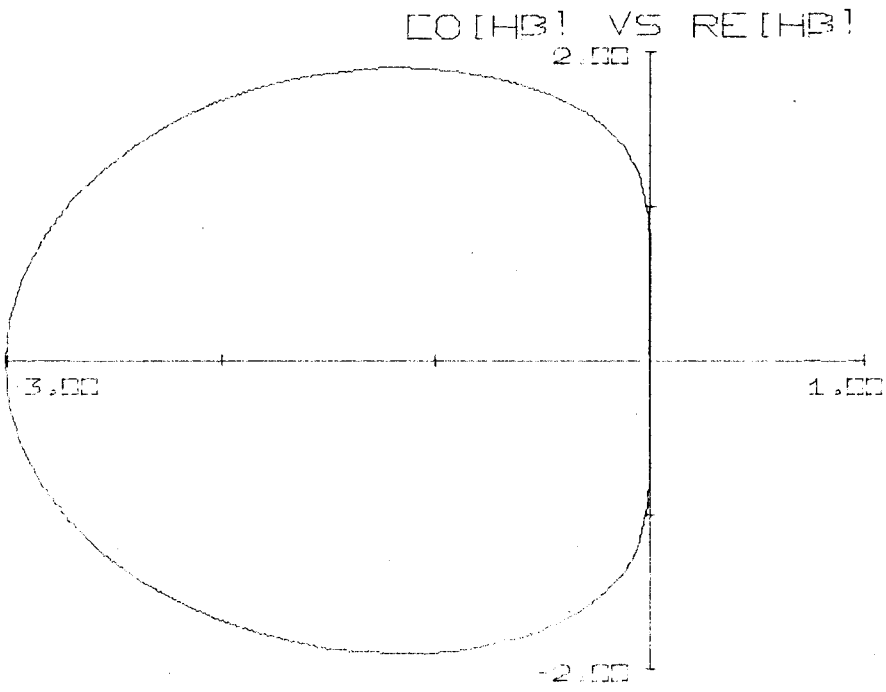


Figure 11. The stability region for the fourth order Adams corrector formula. The region is plotted in the complex $\bar{h}$ plane.

The correction values are obtained by noting that the predicted value p_{i+1} from Equation (6) is in error by $\frac{251}{720} h^5 f^{(4)}(\xi)$, that the corrected value c_{i+1} is in error by $-\frac{19}{720} h^5 f^{(4)}(\xi)$, and that if $f^{(4)}$ is a constant, then

$$p_{i+1} - c_{i+1} = \frac{270}{720} h^5 f^{(4)}$$

Subtracting $\frac{251}{270}(p_i - c_i)$ from p_{i+1} would compensate for the

truncation error in the predictor, and adding $\frac{19}{270}(p_{i+1}-c_{i+1})$ to c_{i+1} would compensate for the truncation error in the corrector. The equations for this method are:

Predict: $$p_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}) \tag{10}$$

Modify: $$m_{i+1} = p_{i+1} - \frac{251}{270}(p_i - c_i) + \frac{95}{288}h^6 f^{(5)}(\xi) \tag{11}$$

Correct: $$c_{i+1} = y_i + \frac{h}{24}(9m'_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}) \tag{12}$$

Modify: $$y_{i+1} = c_{i+1} + \frac{19}{270}(p_{i+1} - c_{i+1}) - \frac{3}{160}h^6 f^{(5)}(\xi) \tag{13}$$

It can be seen that a fifth order method has been achieved with very little added calculation. However, the stability of the method must be investigated before any conclusion can be drawn.

Chase has shown that it is not sufficient to look at the stability of the corrector alone for methods which do not iterate to convergence and which do not have $\bar{h}$ sufficiently close to zero. The discussion below of the stability for this method follows that done by Chase for Hamming's modified method (2, p. 457-468).

Let z_i be the true solution of the problem being solved and define

$$\beta_i = z_i - p_i, \quad \eta_i = z_i - m_i, \quad \gamma_i = z_i - c_i, \quad \text{and} \quad \epsilon_i = z_i - y_i.$$

Subtracting Equations (10)-(13) from the true solution z_i, applying

the Mean Value Theorem, and assuming that $\partial f/\partial y$ is constant gives the following equations:

$$\beta_{i+1} = \epsilon_i + \frac{\bar{h}}{24}(55\epsilon_i - 59\epsilon_{i-1} + 37\epsilon_{i-2} - 9\epsilon_{i-3}) \tag{14}$$

$$\eta_{i+1} = \beta_{i+1} - \frac{251}{270}(\beta_i - \gamma_i) + E_{i+1} \tag{15}$$

$$\gamma_{i+1} = \epsilon_i + \frac{\bar{h}}{24}(9\eta_{i+1} + 19\epsilon_i - 5\epsilon_{i-1} + \epsilon_{i-2}) \tag{16}$$

$$\epsilon_{i+1} = \gamma_{i+1} + \frac{19}{270}(\beta_{i+1} - \gamma_{i+1}) + \bar{E}_{i+1} \tag{17}$$

where $\bar{h} = h\frac{\partial f}{\partial y}$ and E_{i+1} and $\bar{E}_{i+1}$ are the truncation error terms.

Equations (14)-(17) are a system of difference equations with solutions of the following form:

$$\beta_i = B\rho^i, \quad \eta_i = C\rho^i, \quad \gamma_i = D\rho^i, \text{ and } \epsilon_i = E\rho^i.$$

Substituting these into Equations (14)-(17) gives a set of simultaneous, homogeneous equations with constants B, C, D, and E. If this system of equations is to have a nontrivial solution, it is sufficient for the determinant of the coefficient matrix to be zero (2, p. 457-468). Taking the determinant of the coefficient matrix gives the following fifth degree polynomial in ρ:

$$\rho^5 - \rho^4\left[\frac{2761}{3456}\bar{h}^2 + \frac{287}{180}\bar{h} + 1\right] + \rho^3\left[\frac{4769}{2880}\bar{h}^2 + \frac{103}{144}\bar{h}\right]$$

$$- \rho^2\left[\frac{251}{180}\bar{h}^2 + \frac{53}{360}\bar{h}\right] + \rho\left[\frac{5773}{8640}\bar{h}^2 + \frac{19}{720}\bar{h}\right] - \frac{251}{1920}\bar{h}^2 = 0 \qquad (18)$$

The region of absolute stability can be calculated by solving Equation (18) for $\bar{h}$ and looking at values of $\rho = e^{i\theta}$, $0 \leq \theta < 2\pi$. The region of absolute stability is shown in Figure 12. From the figure, it can be seen that for $\bar{h}$ less than approximately -0.80 the method is unstable. This result can be verified by looking at the problem:

$$y'(t) = -100y + 100, \quad y(0) = 0$$

with solution

$$y(t) = 1 - e^{-100t}$$

Figures 13 through 16 show choices of $\bar{h}$ just outside this region and the resulting error growth. Figures 17 through 20 show choices of $\bar{h}$ inside the stable region and the resulting damping out of the errors.

The stability region for the modified method seems to be unduly small when compared to that of the iterated method using the same predictor and corrector. However, the modified method is exact for fifth degree polynomials and should be compared with a method of the same order.

The modified method applies the predictor and then the corrector only once. For comparison, a fifth order Adams predictor and

corrector are used. The predictor and corrector are applied in the following way:

Predict:

$$y_{i+1} = y_i + \frac{h}{720}(1901f_i - 2774f_{i-1} + 2616f_{i-2} - 1274f_{i-3} + 251f_{i-4})$$

Correct:

$$y_{i+1} = y_i + \frac{h}{720}(251f_{i+1} + 646f_i - 264f_{i-1} + 106f_{i-2} - 19f_{i-3})$$

i.e., the corrector is applied only once. The characteristic equation for this method was calculated in the same manner as for the modified method above. Figure 21 shows the resulting stability region for the fifth order predictor-corrector method. The stability region for the modified fourth order method shown in Figure 12 is a little smaller than that shown in Figure 21 for the fifth order predictor corrector method, indicating that the modification adds instability.

Throughout the calculation of the solution for a system of differential equations, the stepwidth may be allowed to change by halving and doubling to keep the local truncation error from becoming too large and swamping the true solution or from becoming too small, requiring unnecessary computation. The user may also have the stepwidth fixed throughout the calculation of a solution. This may be done if the stability of a problem is to be examined, or as in the example above, the stability of the method is to be examined.

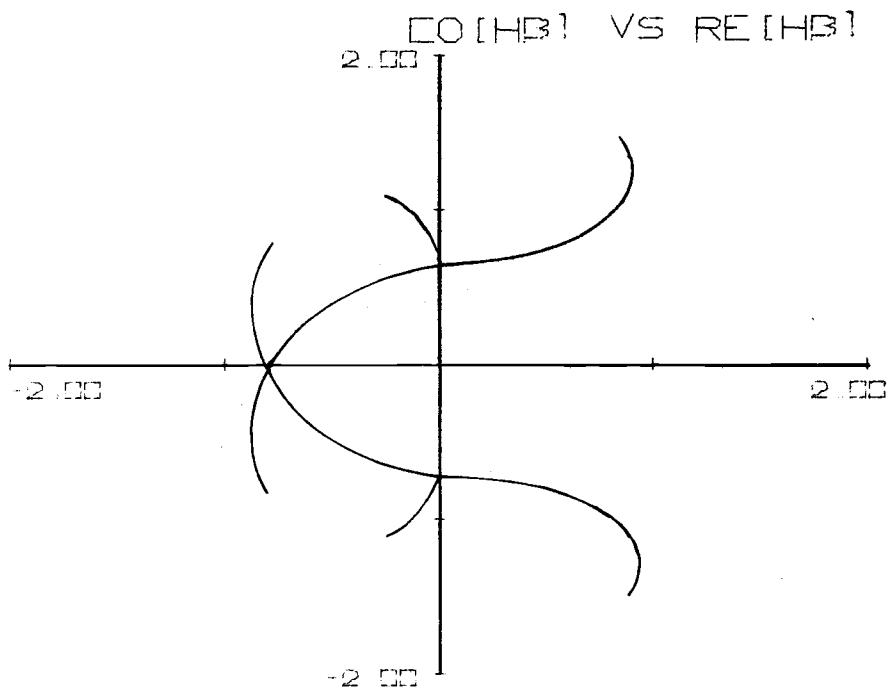


Figure 12. The stability region for the modified predictor-corrector method. The region is plotted in the complex $\bar{h}$ plane.

If the condition

$$\frac{|p_{i+1}-c_{i+1}|}{|c_{i+1}|} \geq 10^{-5}$$

is satisfied while a point is being calculated, the stepwidth is halved and the point being calculated is not used. When the stepwidth is halved, two intermediate points $y_{i-1/2}$ and $y_{i-3/2}$ are needed. They are calculated using the interpolation formulas below:

$$y_{i-1/2} = \frac{1}{256}(80y_i+135y_{i-1}+40y_{i-2}+y_{i-3}-h(5f_i+90f_{i-1}-15f_{i-2}))$$

$$y_{i-3/2} = \frac{1}{256}(12y_i+135y_{i-1}+108y_{i-2}+y_{i-3}-h(3f_i+54f_{i-1}-27f_{i-2}))$$

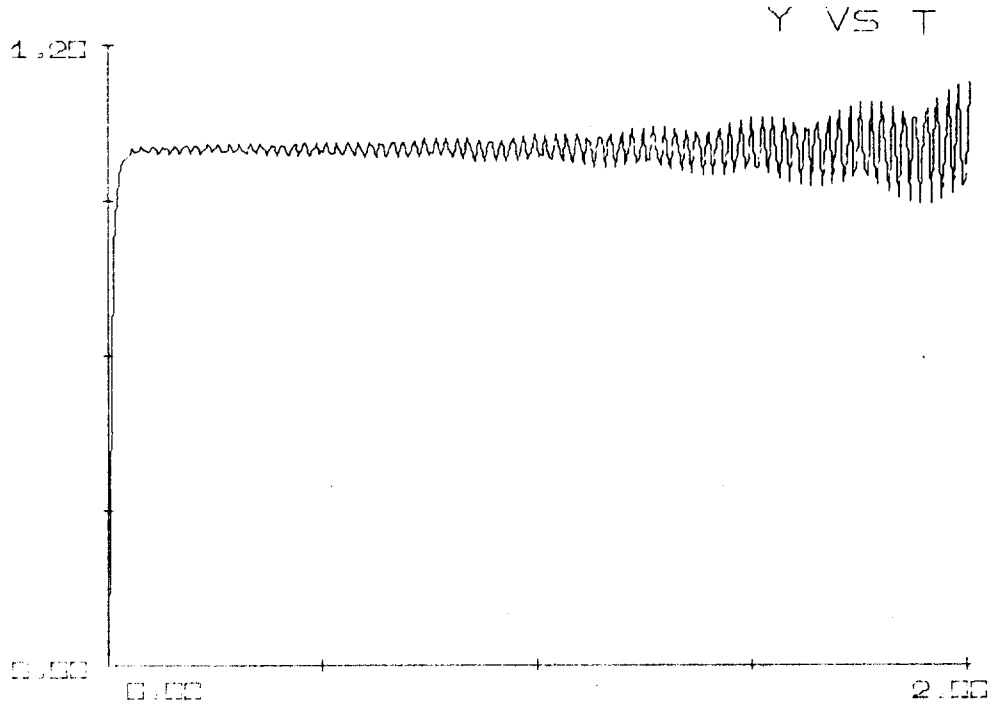


Figure 13. The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.813$.

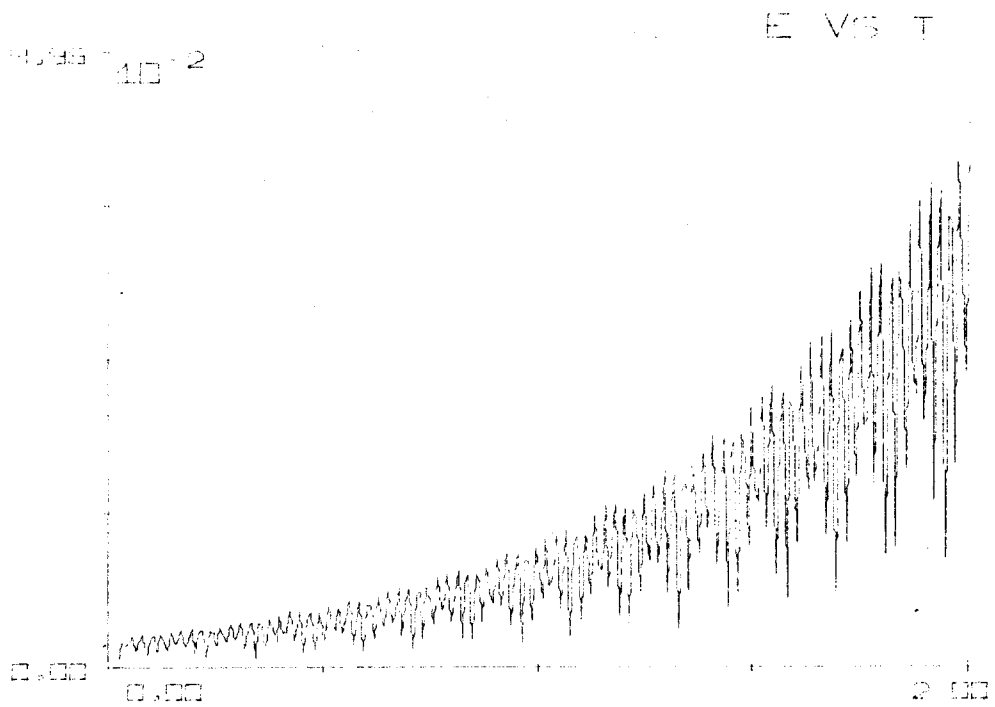


Figure 14. The error bound for the calculated solution shown in Figure 13.

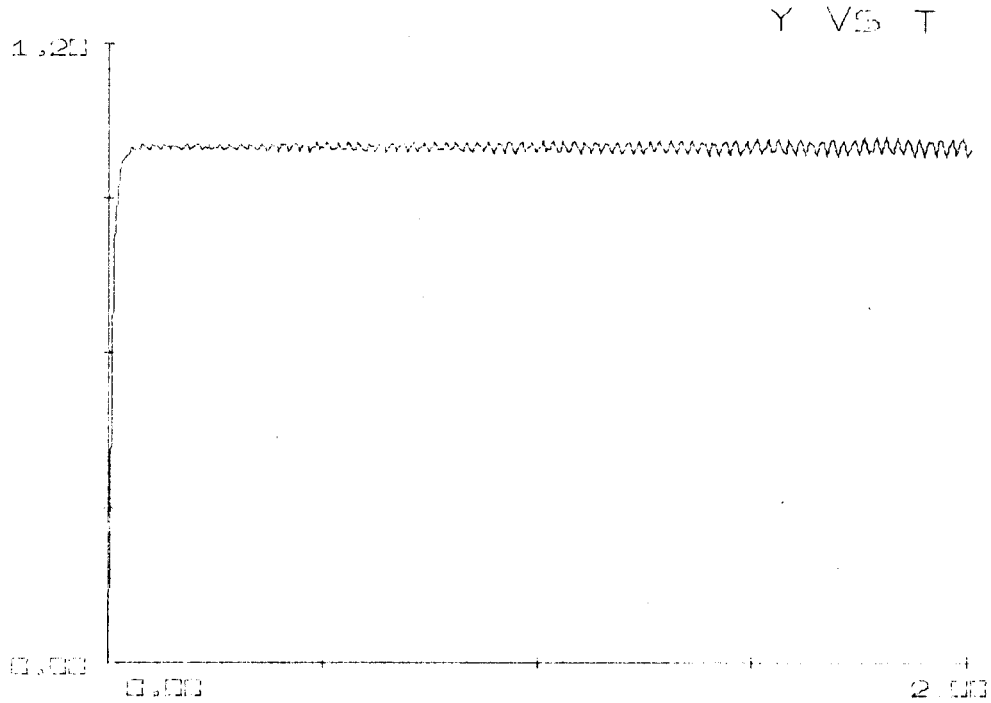


Figure 15. The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.806$.

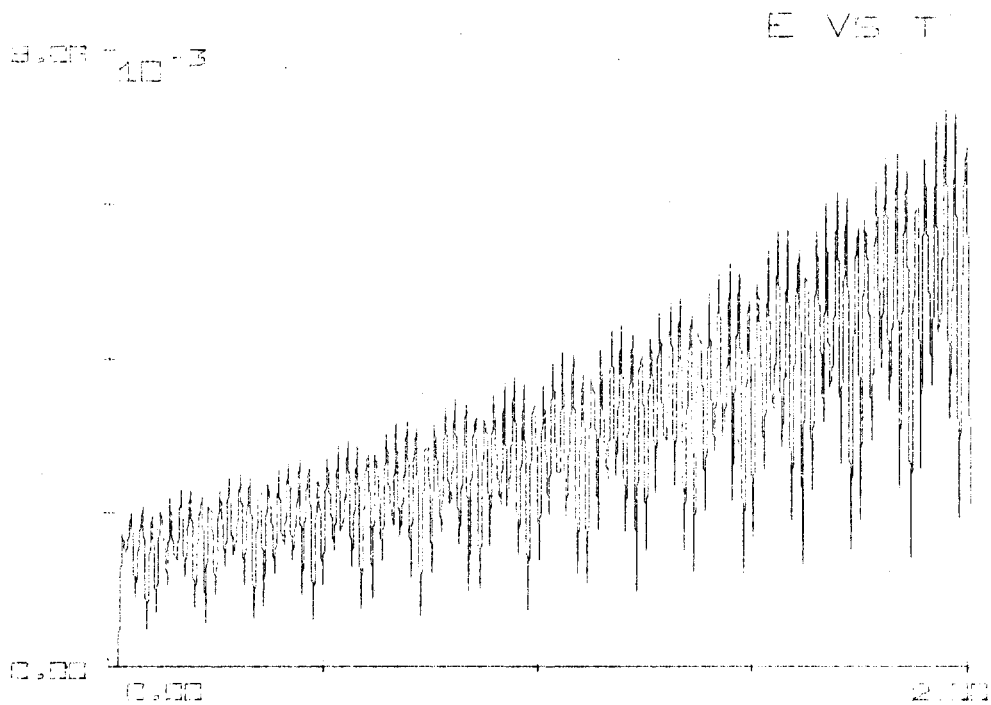


Figure 16. The error bound for the calculated solution shown in Figure 15.

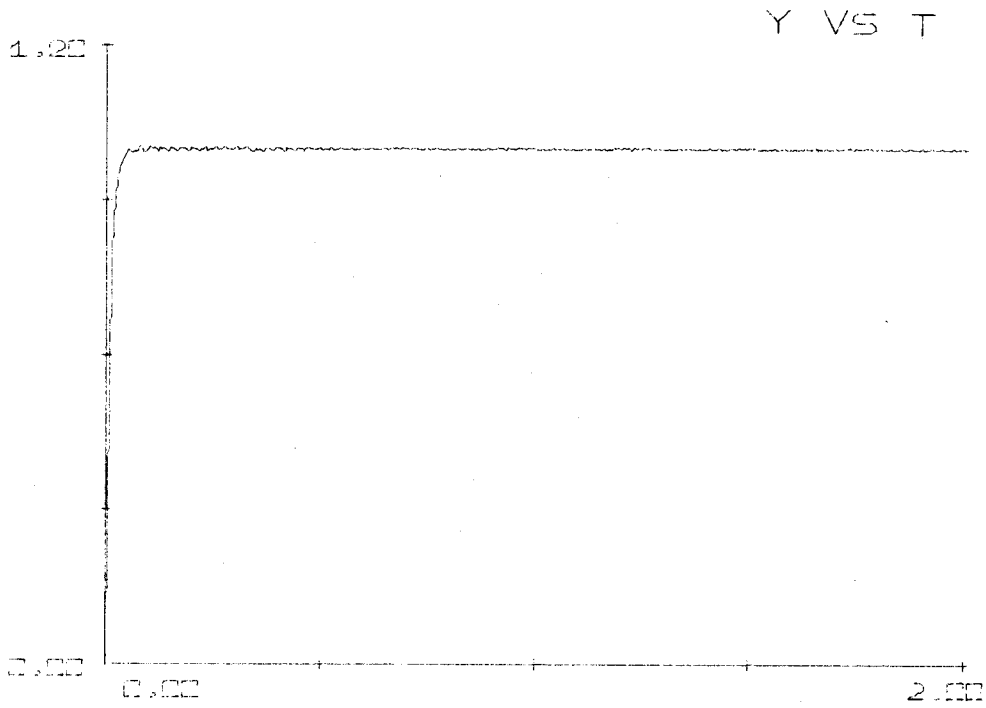


Figure 17. The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\bar{h} = -0.794$.

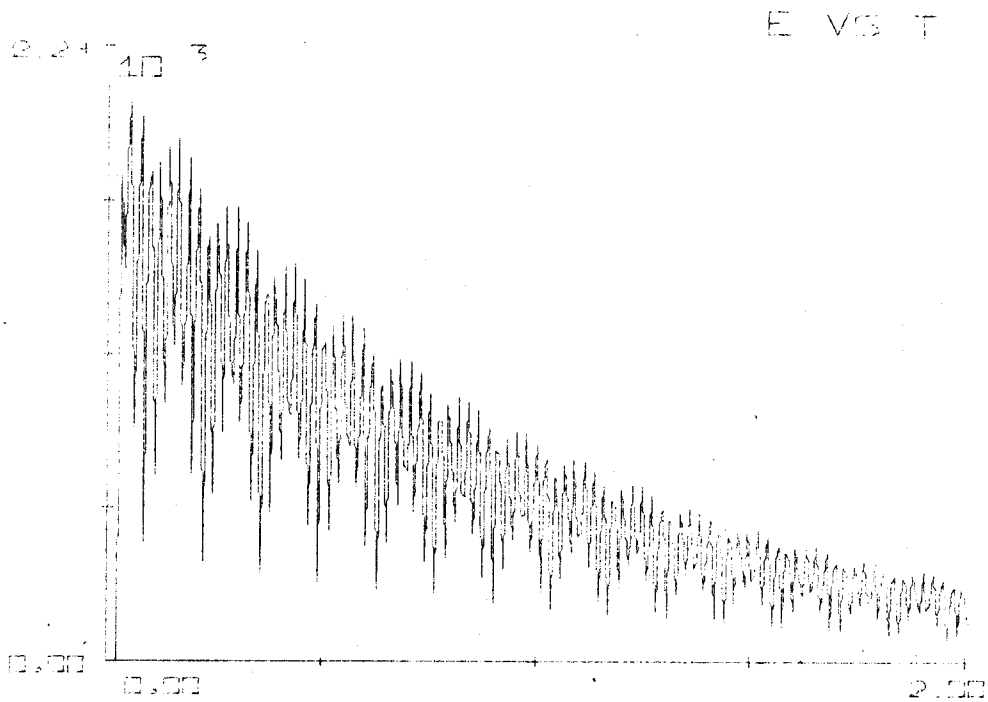


Figure 18. The error bound for the calculated solution shown in Figure 17.

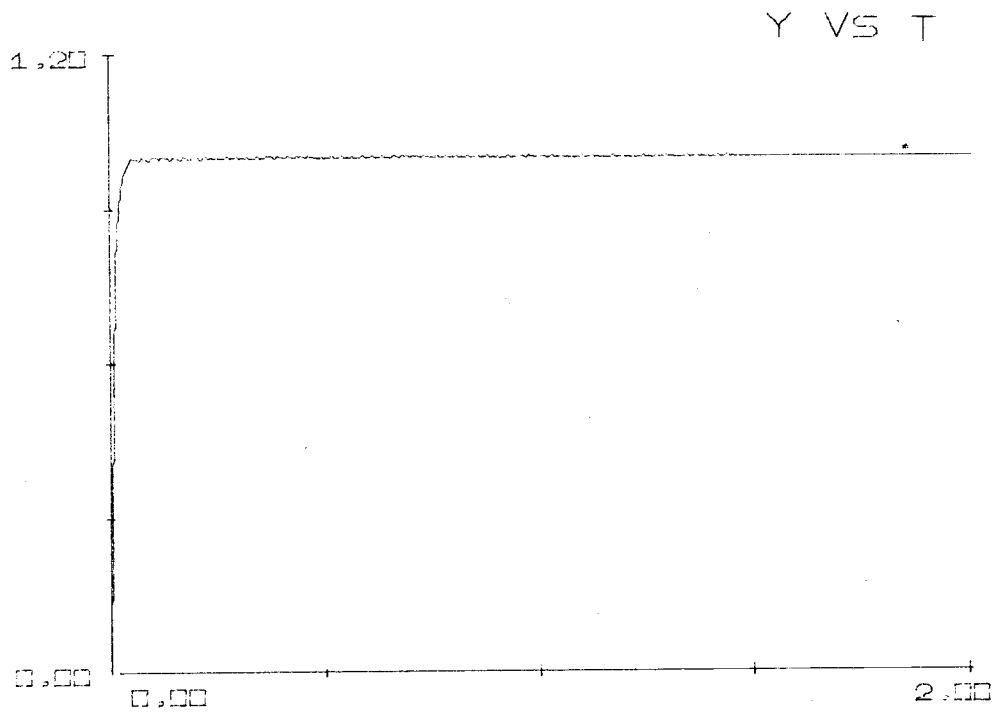


Figure 19. The calculated solution for the problem $y'(t) = -100y + 100$ using the modified method with $\overline{h} = -0.787$.

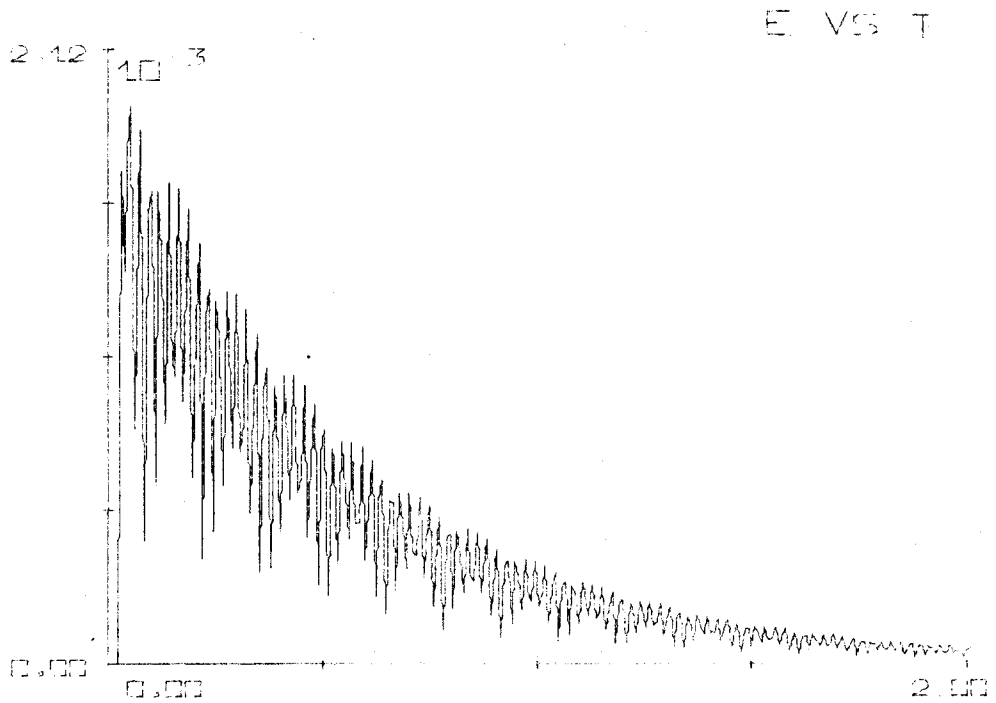


Figure 20. The error bound for the calculated solution shown in Figure 19.

After these values have been inserted, the calculations are resumed with $h = h/2$. If, during the calculations, the stepwidth becomes 1/1024th of the initial stepwidth, the calculations are stopped and an error message is printed.

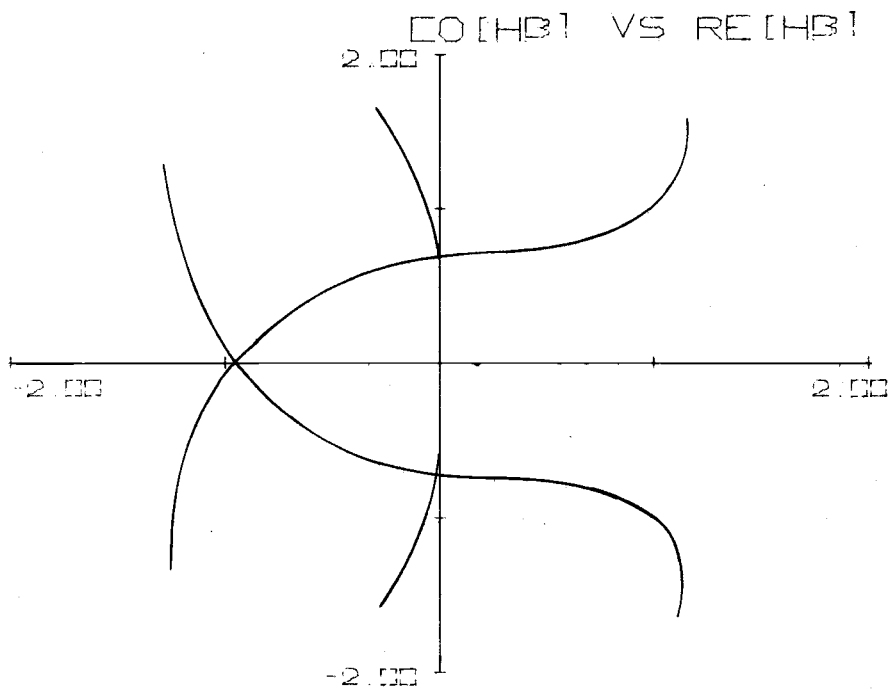


Figure 21. The stability region for the fifth order Adams predictor-corrector method with the corrector being applied only once.

The stepwidth is doubled when

$$\left\| \frac{p_{i+1} - c_{i+1}}{c_{i+1}} \right\|_{\infty} < 10^{-8}$$

to keep the local truncation error from becoming too small. Doubling

is quite easy, since the back values y_{i-4} and y_{i-5} are saved. These values are moved into the appropriate arrays, the stepwidth set to 2h, and the calculations continued. After the stepwidth is doubled, doubling is not allowed for the two successive steps.

The automatic changing of the stepwidth makes the calculation of a solution of differential equations more economical than using a fixed stepwidth. However, a problem arises when the solutions of a system of differential equations are used in other functions not in the equation set. This occurs because the solutions are not available at equally spaced points. A general interpolation formula has been incorporated in the differential equation solving routine so that the points requested are available.

The general form of the interpolation polynomial is:

$$y_{i-\delta} = w_0 y_i + w_1 y_{i-1} + w_2 y_{i-2} + w_3 y_{i-3} + h(w_4 f_i + w_5 f_{i-1} + w_6 f_{i-2})) \quad (19)$$

where the w's are calculated by the matrix multiplication (7, p. 128-130):

$$W = A^{-1} \cdot X$$

where

$$W = (w_0, w_1, w_2, w_3, w_4, w_5, w_6)$$

and

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ X_1 & X_2 & X_3 & X_4 & 1 & 1 & 1 \\ \vdots & & & & & & \\ X_1^6 & X_2^6 & X_3^6 & X_4^6 & 6X_1^5 & 6X_2^5 & 6X_3^5 \end{pmatrix}$$

and

$$X = (1, \delta, \delta^2, \delta^3, \delta^4, \delta^5, \delta^6) \qquad \delta \in [0,3]$$

This feature allows the differential equations to be solved with the stepwidth, which keeps the truncation error within certain bounds and yet allows the solutions to be used as any other function.

It is quite important that a program to solve differential equations have some means for computing a bound for the total or propagated error in solving a differential equation. The differential equation solving routine in this system does allow the user to have a bound on the total error computed. The following formula is used to calculate an approximate bound on the propagated error:[3]

$$E_{i+1} \lesssim |1+hJ_{i+1}| E_i + \| T_{i+1} \|_\infty + \| R_{i+1} \|_\infty \qquad (20)$$

where T_{i+1} is the truncation error vector, R_{i+1} is the roundoff error vector, and J_{i+1} is the largest eigenvalue of the matrix of partial derivatives for the system of equations at the i+1st point.

---

[3] Davis, J., Associate Professor of Math., O.S.U. Private communication, 1972.

Since the calculation of the terms J_{i+1} and R_{i+1} are difficult and depend on the particular problem, the program allows the user to specify an expression or numerical value for these quantities. If the user has specified that the error bound is to be calculated, these quantities are requested along with the initial conditions for the problem. Figure 22 shows a differential equation being entered and these quantities being specified.

```
Y'(T) = -100*Y+ 100$        (equation passed to MINITRAN)
    Y = 0
  ERR = -100                (largest eigenvalue)
 RERR = -0                  (roundoff error term)
```

Figure 22. Quantities for the largest eigenvalue and the roundoff error are requested with the initial conditions. A -0 for the roundoff error causes the quantity $10^{-10} \cdot \|y\|_\infty$ to be used.

The truncation error term is calculated using the difference between the predicted and corrected values. For the iterated method the value used is:

$$T_{i+1} = \frac{19}{270}(p_{i+1}-c_{i+1}) \simeq \frac{19}{720} h^5 f^{(4)}(\xi)$$

and for the modified method the value used is:

$$T_{i+1} = \frac{27}{502}(m_{i+1}-y_{i+1}) \simeq \frac{3}{160} h^6 f^{(5)}(\xi) .$$

A special function ERR has been provided as a standard function to return the values calculated for the total error. This function was used to get the values for the graphs of the error shown in Figures 14 through 20.

Numerical Differentiation

The first and second derivatives of a function specified by a formula are calculated using first and second difference formulas (13, p. 293). The formula used for the first derivative is:

$$y'(x) = \frac{y(x+h)-y(x-h)}{2h} - \frac{h}{6} y^{(3)}(\xi) \qquad (21)$$

where h is the independent variable increment. Equation (21) is used with two stepwidths h and 2h to get estimates $\bar{y}'(x)$ and $\bar{\bar{y}}'(x)$. The two estimates satisfy the following relations:

$$y'(x) = \bar{y}'(x) + O(h^2) \qquad (22)$$

$$y'(x) \simeq \bar{\bar{y}}'(x) + 4O(h^2) \qquad (23)$$

Eliminating the $O(h^2)$ term from (22) and (23) gives the following result, which is a better approximation:

$$y'(x) \simeq \bar{y}'(x) + \frac{\bar{y}'(x)-\bar{\bar{y}}'(x)}{3}$$

The second derivative is calculated in a similar way. Two

estimates $\bar{y}''$ and $\bar{\bar{y}}''(x)$ are calculated using the following formula:

$$y''(x) = \frac{y(x+h)-2y(x)+y(x-h)}{h^2} - \frac{h^2}{12} y^{(4)}(\xi)$$

Again, there are two approximations for the same quantity which satisfy:

$$y''(x) = \bar{y}''(x) + O(h^2) \tag{24}$$

$$y''(x) \simeq \bar{\bar{y}}''(x) + 4O(h^2) \tag{25}$$

Eliminating $O(h^2)$ from Equations (24) and (25) gives the result:

$$y''(x) \simeq \bar{y}''(x) + \frac{\bar{y}''(x)-\bar{\bar{y}}''(x)}{3}$$

In both the calculation of $y'(x)$ and $y''(x)$, the calculations are done with stepwidths h and $2h$, and the resulting estimates are used to get an improved result. This technique is called Richardson's extrapolation and is described in several textbooks (13, p. 374; 9, p. 239).

Numerical Integration

The system-supplied function INT calculates the definite integral of a function specified by a formula. The integral is calculated using Simpson's Rule:

$$\int_a^b f(x)dx = \frac{h}{3}(f_0+4f_1+2f_2+\ldots+4f_{n-1}+f_n) - \frac{nh^5 f^{(4)}(\xi)}{180} \qquad (21)$$

where n and h are specified by the user in the call to INT. Two intervals h and $h/2$ are used to calculate estimates I_1 and I_2 of the integral and to get the following relations:

$$I = I_1 + O(h^4) \qquad (26)$$

$$I \simeq I_2 + \frac{O(h^4)}{16} \qquad (27)$$

Eliminating the $O(h^4)$ term from Equations (26) and (27) gives the following result (17, p. 245):

$$I \simeq I_2 + \frac{I_2-I_1}{15}$$

BIBLIOGRAPHY

1. Arden, B.W., B.A. Galler and R.M. Graham. An algorithm for translating Boolean expressions. Journal of the Association for Computing Machinery 9:222-239. 1962.

2. Chase, P.E. Stability properties of predictor-corrector methods for ordinary differential equations. Journal of the Association for Computing Machinery 9:457-468. 1962.

3. Crane, R.L. and R.J. Lambert. Stability of a generalized corrector formula. Journal of the Association for Computing Machinery 9:104-117. 1962.

4. Gear, G.W. The automatic integration of stiff ordinary differential equations. In: Proceedings of International Federation for Information Processing, Edinburgh, 1968. North-Holland Publishing Company, Amsterdam, 1969. p. 187-193.

5. Gear, G.W. Numerical initial value problems in ordinary differential equations. Englewood Cliffs, N.J., Prentice Hall, 1971. 253 p.

6. Gear, G.W. Numerical solutions of ordinary differential equations at a remote terminal. In: Proceedings for the Association for Computing Machinery 21st National Conference. MDI Publns., Wayne, Pa., 1966. p. 43-49.

7. Hamming, R.W. Numerical methods for scientists and engineers. New York, McGraw-Hill, 1962. 411 p.

8. Hamming, R.W. Stable predictor-corrector methods for ordinary differential equations. Journal of the Association for Computing Machinery 6:37-47. 1959.

9. Henrici, P. Elements of numerical analysis. New York, John Wiley and Sons, 1967. 336 p.

10. Hildebrand, F.B. Introduction to numerical analysis. New York, McGraw-Hill, 1956. 511 p.

11. Hull, T.E. The numerical integration of ordinary differential equations. In: Proceedings of International Federation for Information Processing, Edinburgh, 1968. North-Holland Publishing Company, Amsterdam, 1969. p. 40-53.

12. Hull, T.E. and A.L. Creemer. Efficiency of predictor-corrector procedures. Journal of the Association for Computing Machinery 10:291-301. 1963.

13. Isaccson, E. and H.B. Keller. Analysis of numerical methods. New York, John Wiley and Sons, 1966. 541 p.

14. Kaplow, R., J. Brackett and S. Strong. Man-machine communication in on-line mathematical analysis. In: Proceedings of American Federation of Information Processing 1966 Fall Joint Computer Conference. Vol. 29, Spartan Books, New York, 1966. p. 465-477.

15. Krogh, F.T. Predictor-corrector methods of high order with improved stability characteristics. Journal of the Association for Computing Machinery 13:374-385. 1966.

16. Milne, W.E. Numerical solutions of differential equations. New York, John Wiley and Sons, 1953. 275 p.

17. Ralston, A. and H.S. Wilf. Mathematical methods for digital computers. New York, John Wiley and Sons, 1966. 293 p.

18. Rice, J.R. and S. Rosen. NAPSS - a numerical analysis problem solving system. In: Proceedings of the Association for Computing Machinery 21st National Conference, MDI Publns., Wayne, Pa. p. 51-56.

19. Shaw, B. Modified multistep methods based on a nonpolynomial interpolant. Journal of the Association for Computing Machinery, 14:143-154. 1967.

20. Smith, L.B. A survey of interactive graphical systems for mathematics. Computing Surveys 2, no. 4:261-301. Dec. 1970.

APPENDICES

APPENDIX A

Modified Backus Normal Form Specification of the Language

<function specification > :: =
    <function definition > $ | <function definition > $
    <function specification >

<function definition > :: =
    <function identification > = <arithmetic expression >

<function identification > :: =
    <function identifier > (<formal parameter list >) |
    <function identifier > <differential equation identifier >
    (<formal parameter list >)

<function identifier > :: =

$$\text{<alpha character>} \int_0^3 \text{<alphanumeric character>}$$

<differential equation indicator > :: =
    ' | " | <differential equation indicator > ' |
    <differential equation indicator > "

<formal parameter list > :: =
    <parameter > | <formal parameter list > , <parameter >

<arithmetic expression > :: =
    <simple arithmetic expression > | <conditional statement >

<simple arithmetic expression > :: =
    <term > | <simple arithmetic expression > <add operator > <term>

<term > :: =
    <factor > | <term > <multiply operator > <term > |
    <unary operator > <term >

<factor > :: =
    <primary > | <factor > <power operator > <primary > | <factor >
    <power operator > <unary operator > <primary >

<primary> :: =
    <unsigned number> | <parameter> | <function designator> |
    (<arithmetic expression>)

<conditional statement> :: =
    <simple arithmetic expression> IF <relational> |
    <conditional statement> <else operation>
    <conditional statement> | <conditional statement>
    <else operation> <simple arithmetic expression>

<relational> :: =
    <simple arithmetic expression> <relation>
    <simple arithmetic expression> | <simple arithmetic expression>
    <relation> <simple arithmetic expression> <relation>
    <simple arithmetic expression>

<parameter> :: =
    <alpha character> $\int_0^1$ <alphanumeric character>

<function designator> :: =
    SIN | COS | . . . | <user defined function>

<alphanumeric character> :: =
    <alpha character> | <numeric character>

<unsigned number> :: =
    <integer> | . <integer> | <integer> . | <integer> . <integer> |
    <integer> E <exponent> | <integer> . E <exponent> | . <integer>
    E <exponent> | <integer> . <integer> E <exponent>

<exponent> :: =
    <integer> | + <integer> | - <integer>

<integer> :: =
    <numeric character> | <integer> <numeric character>

<add operator> :: =
    + | -

<unary operator> :: =
    + | -

<multiply operator> :: =
    * | /

<power operator> :: =
    ↑ | **

<relation> :: =
    < | > | # | = | < = | > =

<else operation> :: =
    ELSE | ;

<alpha character> :: =
    A | B | C ... Y | Z

<numeric character> :: =
    0 | 1 | 2 ... 8 | 9

APPENDIX B

Calling Sequences for System Routines

The system described in this paper has been divided into two routines which must be called by a program using the system. The first routine is MINITRAN which translates the function definitions into executable machine code and generates the tables and maps necessary to evaluate the functions. The second routine FUN is used to evaluate the functions that have been defined using MINITRAN. These routines have been written as FORTRAN compatible subroutines and the discussion to follow will assume they are being called from a FORTRAN main program.

The COMMON/DATA area is used by MINITRAN and FUN to communicate with the main program. The information contained in the COMMON/DATA area is described below in the order in which it must appear. The type (real or integer) of the variables is determined by the standard FORTRAN variable naming conventions, i.e., variables whose first character is I, J, K, L, M, or N are integer and variables whose first character is any other letter are real. The entries in the COMMON/DATA area are:

ITERFLG - determines which predictor-corrector scheme is to be used when solutions to differential equations are calculated.

0 - Iterated Method

A fourth order Adams corrector is iterated until the normalized difference between successive corrected values is less than 10^{-10}.

-0 - Modified Predictor-Corrector Method

This method involves adding a correction to the predicted and corrected values at each step to compensate for the truncation error.

IFIXED - determines whether the initial stepwidth h is to be allowed to be doubled and halved by the program or is to remain constant throughout the calculation of the solution to a system of differential equations.

0 - the stepwidth h is to remain fixed throughout the calculations.

-0 - the stepwidth h is to be halved and doubled to keep the local truncation error in the interval $(10^{-5}, 10^{-8})$.

IERROR - the value of this variable determines if the bound for the total error is to be calculated when systems of differential equations are solved.

0 - the error bound will be calculated when a system of differential equations is solved. When a system of differential equations is compiled, the quantities for the largest eigenvalue and the roundoff error used in the calculation of the error bound will be requested along with the initial conditions of the set of equations.

-0 - the error bound will not be calculated when a system of differential equations is solved and the quantities needed for the calculation of the error bound will not be requested.

BUFFER - this array must contain the function definitions as a character string each time MINITRAN is called. A dollar sign ($) must be the last character in the definition of a function specified by a formula. Systems of differential equations are passed to MINITRAN in this array

with and signs (&) separating the individual members of the equation set and with a dollar sign ($) following the last equation in the set. MINITRAN uses this array during compilation of a function.

In addition to the COMMON/DATA area used to pass information from the main program to MINITRAN, the COMMON area is used to store the machine code generated for each function. Hence, a program calling MINITRAN cannot use the COMMON area for program storage.

The calling sequence for MINITRAN is:

CALL MINITRAN(KODE)

where the parameter is:

KODE - indicates whether an error was detected in the function definition. If KODE is zero the function was compiled without errors, and if KODE is nonzero, some error was detected and an appropriate message printed. The error messages printed and a description of each are given below.

The subroutine FUN is called to evaluate a particular function which has been processed by MINITRAN. The routine will handle functions with any number of independent variables. However, it will only increment the first independent variable. The calling sequence for FUN is:

CALL FUN(NAME, NPRIMES, TINT, DELTAT, NPOINTS, VALUES)

where the parameters are:

NAME - the name of the function to be evaluated as four BCD characters. If the function has not been defined an error message is printed.

NPRIMES - the number of primes following the function name, if the function is a solution to a system of differential equations.

TINT - an array containing initial values for the independent variables of the function. For example, if the function F(X, Y, Z) has been defined using MINITRAN, the entries in TINT would be:

```
X = TINT(1)
Y = TINT(2)
Z = TINT(3)
```

DELTAT - the increment for the first independent variable of the function.

NPOINTS - the number of points at which the function is to be evaluated.

VALUES - upon return from FUN this array contains the values calculated for the specified function. The entries in this array correspond to values of the function calculated at the points:

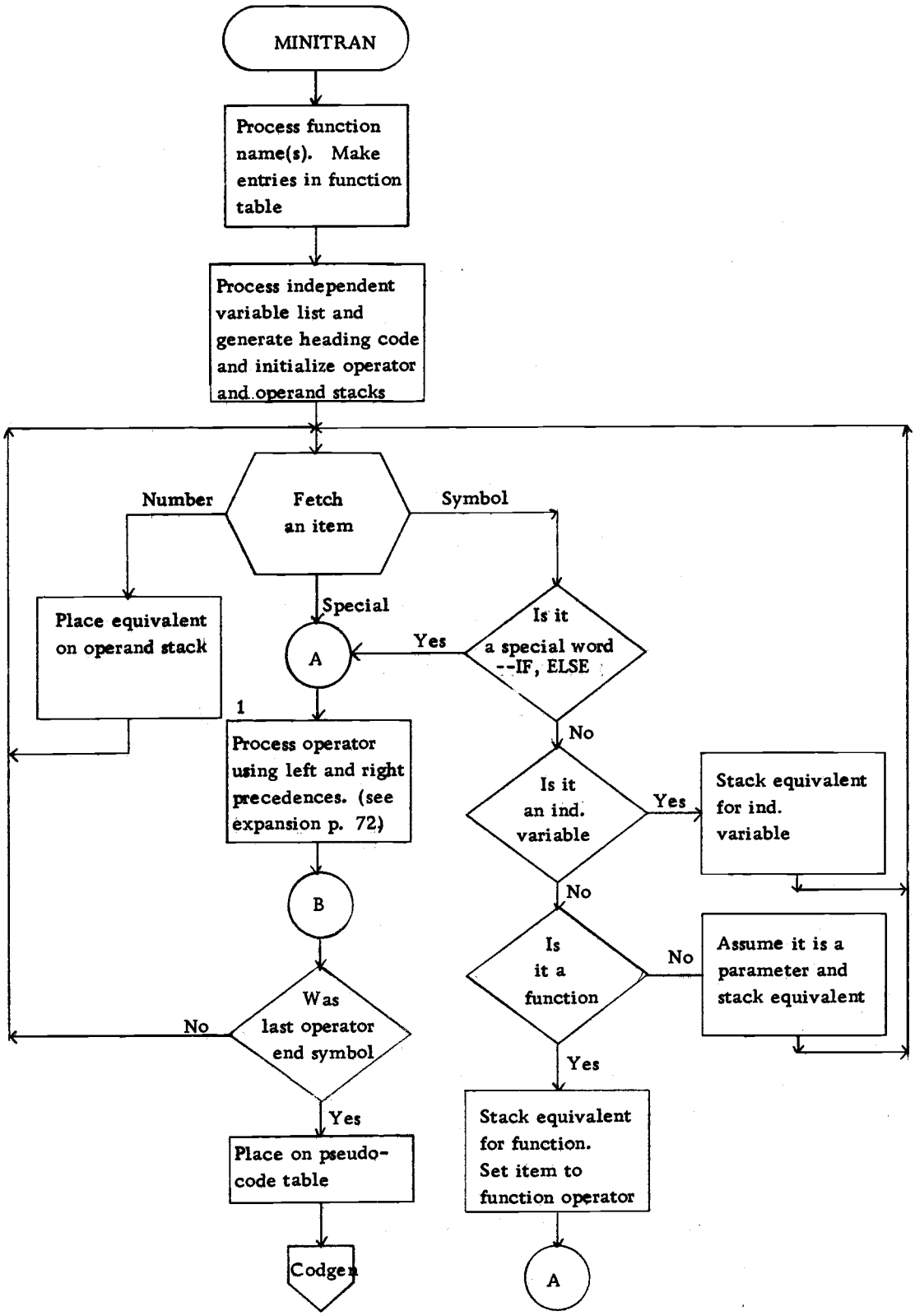
$$t_0, t_0 + \Delta t, \ldots t_0 + (\text{NPOINTS}-1) \cdot \Delta t$$

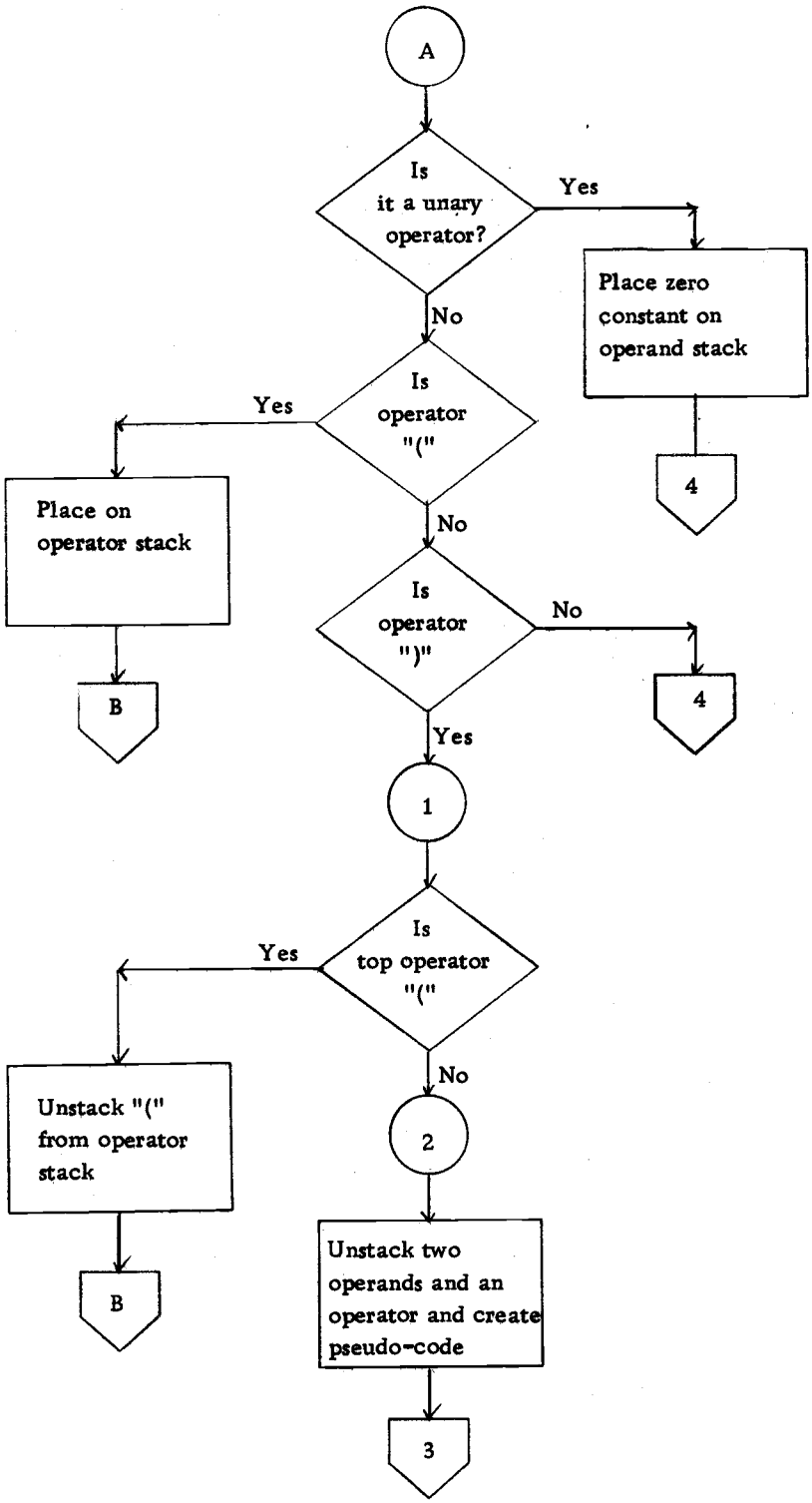
where t_0 is the initial value of the first independent variable and Δt is the independent variable increment DELTAT.

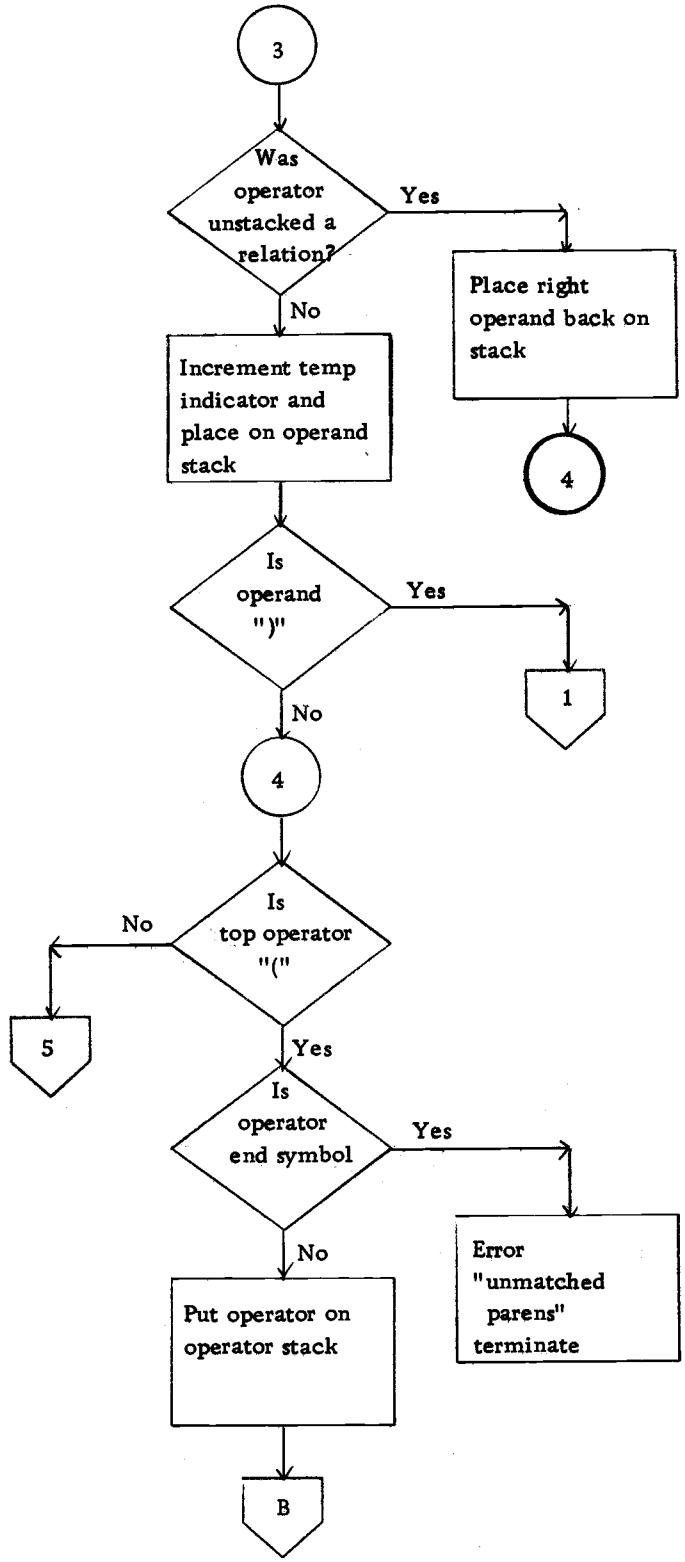
A list of the error messages printed by MINITRAN during compilation is given below:

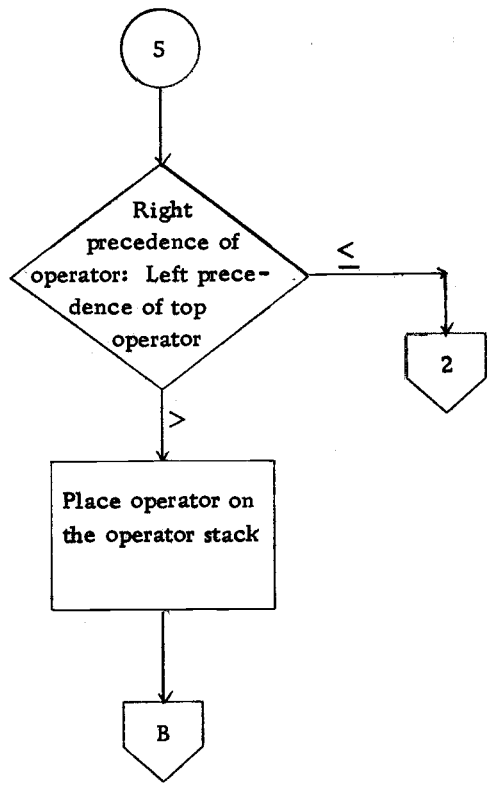
Error Message	Description
TOO MANY FUNCTIONS	More than 15 functions have been defined.
TOO MANY INDEPENDENT VARIABLES	More than 15 different independent variable names have been used.
MEMORY OVERFLOW, RESTART	The available storage for function code has been filled.

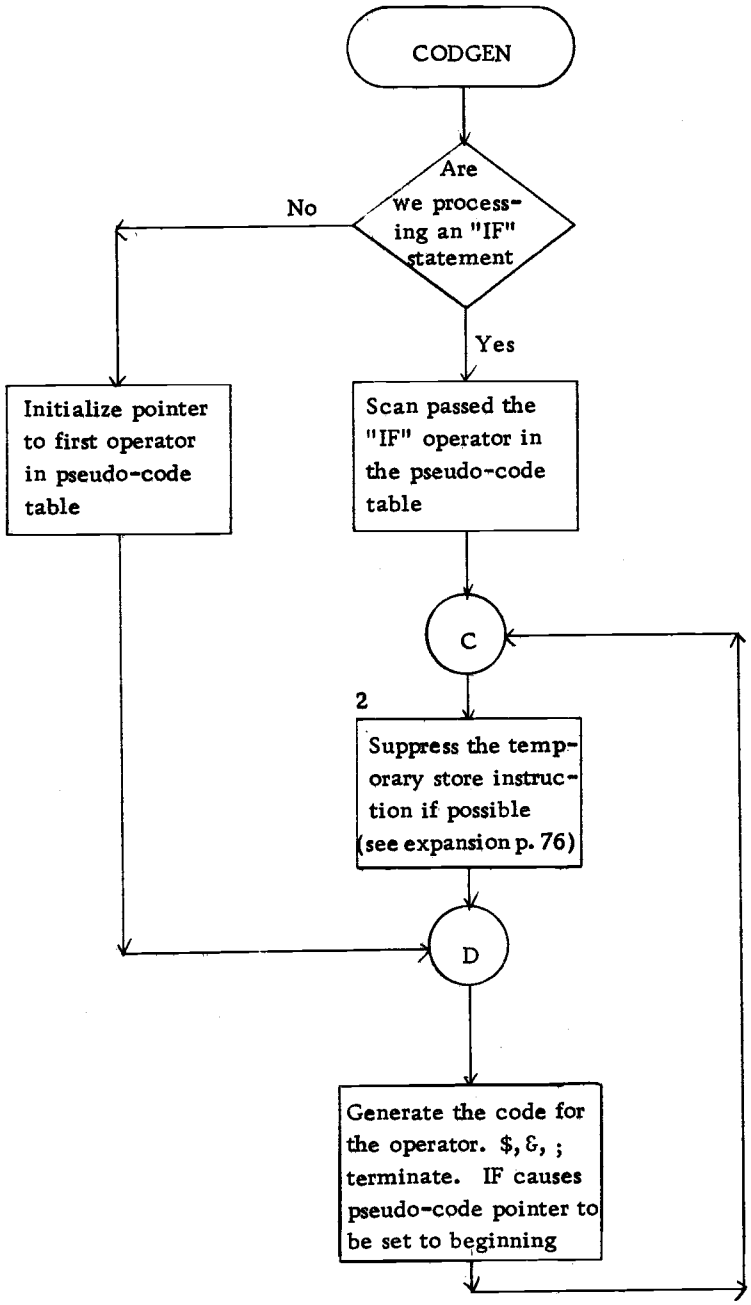
Error Message	Description
EXPRESSION TOO COMPLICATED	The expression contains more than 60 operators.
TOO MANY DIFF. EQUATIONS IN SYSTEM	A system of differential equations has been created in which the number of equations plus the orders is greater than 15.
TOO MANY PARAMETERS	More than 30 parameters have been defined.
ILLEGAL FUNCTION NAME	The function name is not composed of an alphabetic character followed by from 0-3 alphanumeric characters.
FUNCTION ALREADY DEFINED	The function name has been used on a previously entered function.
ILLEGAL PARAMETER ELEMENT	A number or special character appears in the list of independent variables in the function definition.
TWO OPERATORS OR OPERANDS IN A ROW	Two binary operators appear consecutively or an undefined function has been referenced.
UNMATCHED PARENTHESIS	The expression contains an unequal number of right and left parenthesis.
DIFFERENTIATING MORE THAN TWICE	The number of primes on a non-differential equation is more than two.

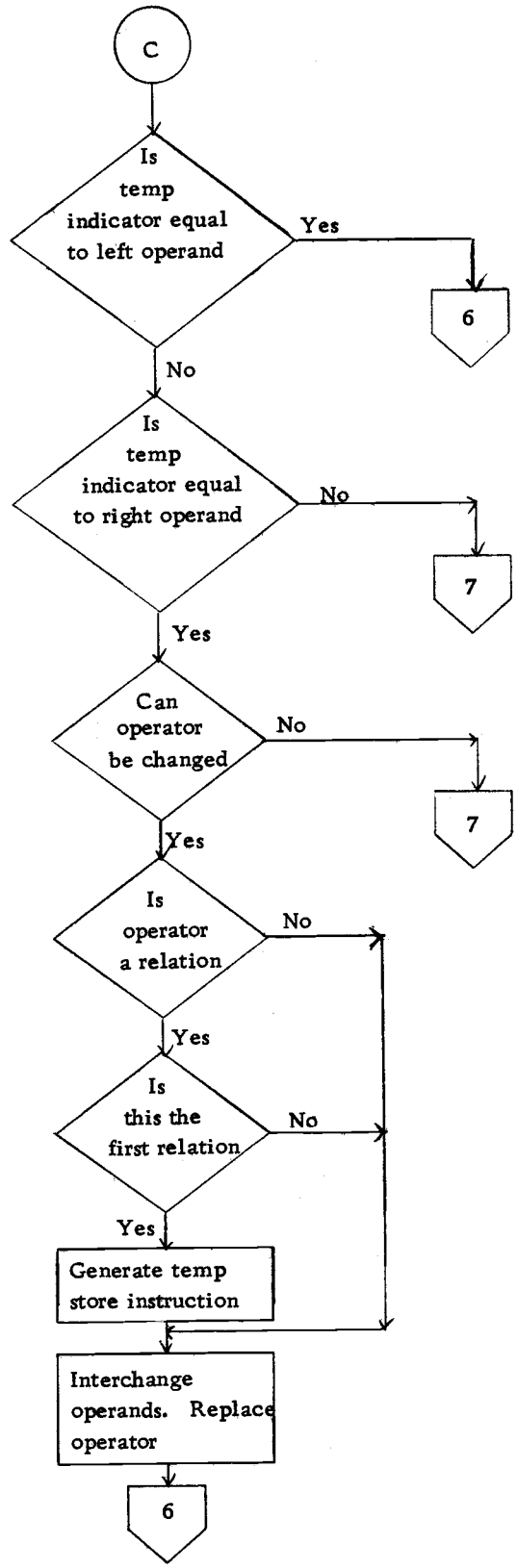

MINITRAN
Process function name(s). Make entries in function table
Process independent variable list and generate heading code and initialize operator and operand stacks
Fetch an item
Number
Symbol
Special
Place equivalent on operand stack
A
Yes
Is it a special word --IF, ELSE
No
1
Process operator using left and right precedences. (see expansion p. 72)
Is it an ind. variable
Yes
Stack equivalent for ind. variable
No
B
Is it a function
No
Assume it is a parameter and stack equivalent
Was last operator end symbol
No
Yes
Stack equivalent for function. Set item to function operator
Yes
Place on pseudo-code table
Codgen
A

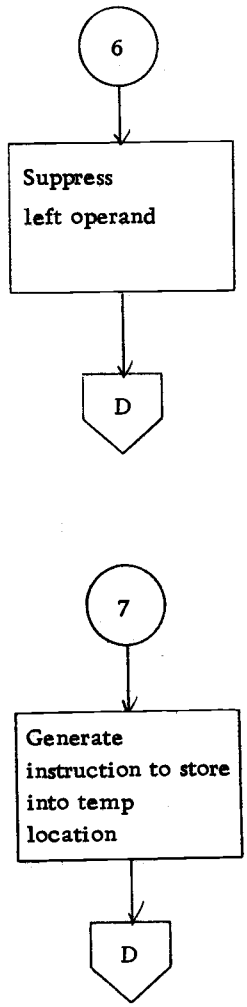

A
Is it a unary operator?
Yes
Place zero constant on operand stack
4
No
Is operator "("
Yes
Place on operator stack
B
No
Is operator ")"
No
4
Yes
1
Is top operator "("
Yes
Unstack "(" from operator stack
B
No
2
Unstack two operands and an operator and create pseudo-code
3

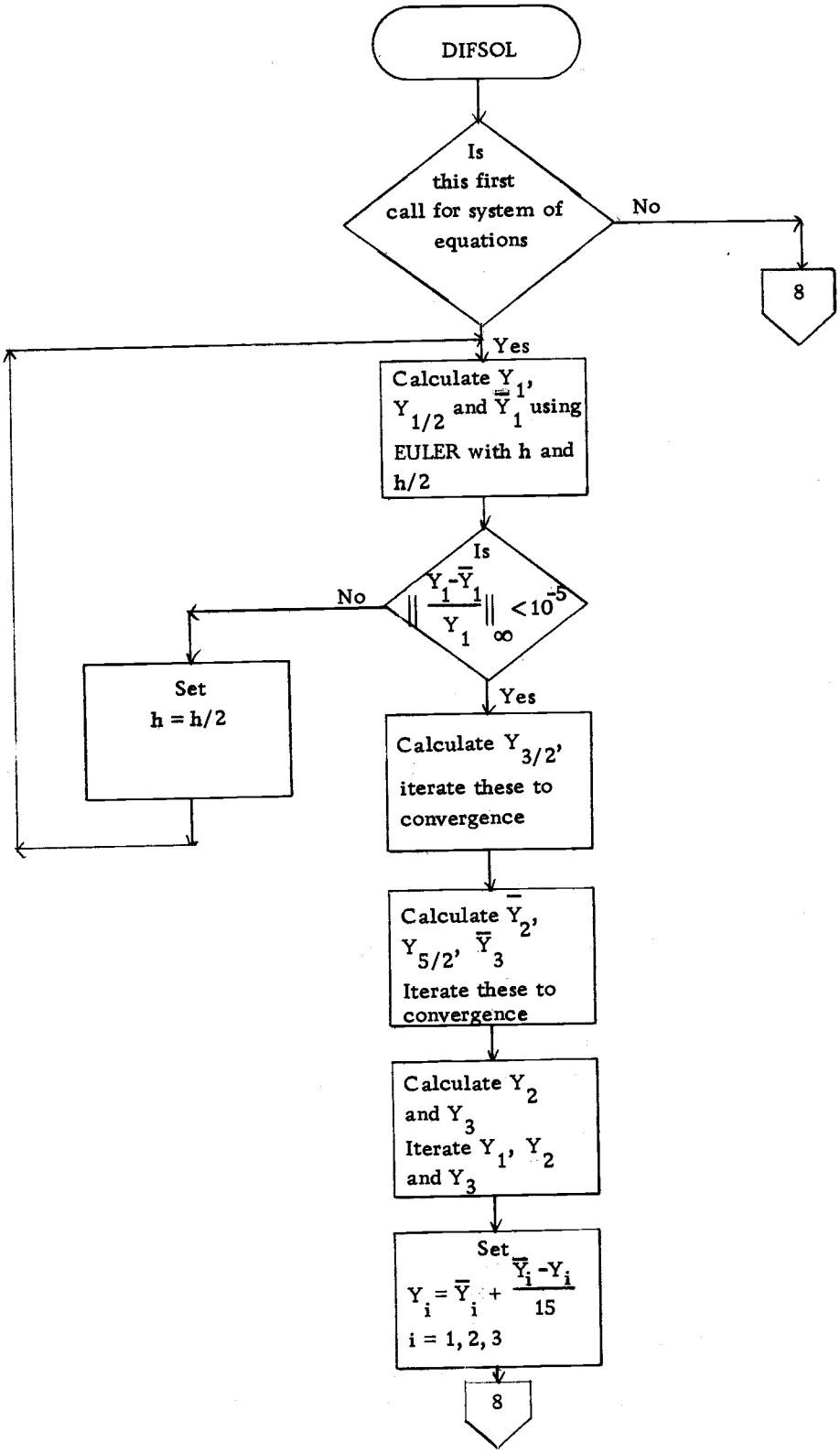

3
Was operator unstacked a relation?
Yes
Place right operand back on stack
4
No
Increment temp indicator and place on operand stack
Is operand ")"
Yes
1
No
4
Is top operator "("
No
5
Yes
Is operator end symbol
Yes
Error "unmatched parens" terminate
No
Put operator on operator stack
B

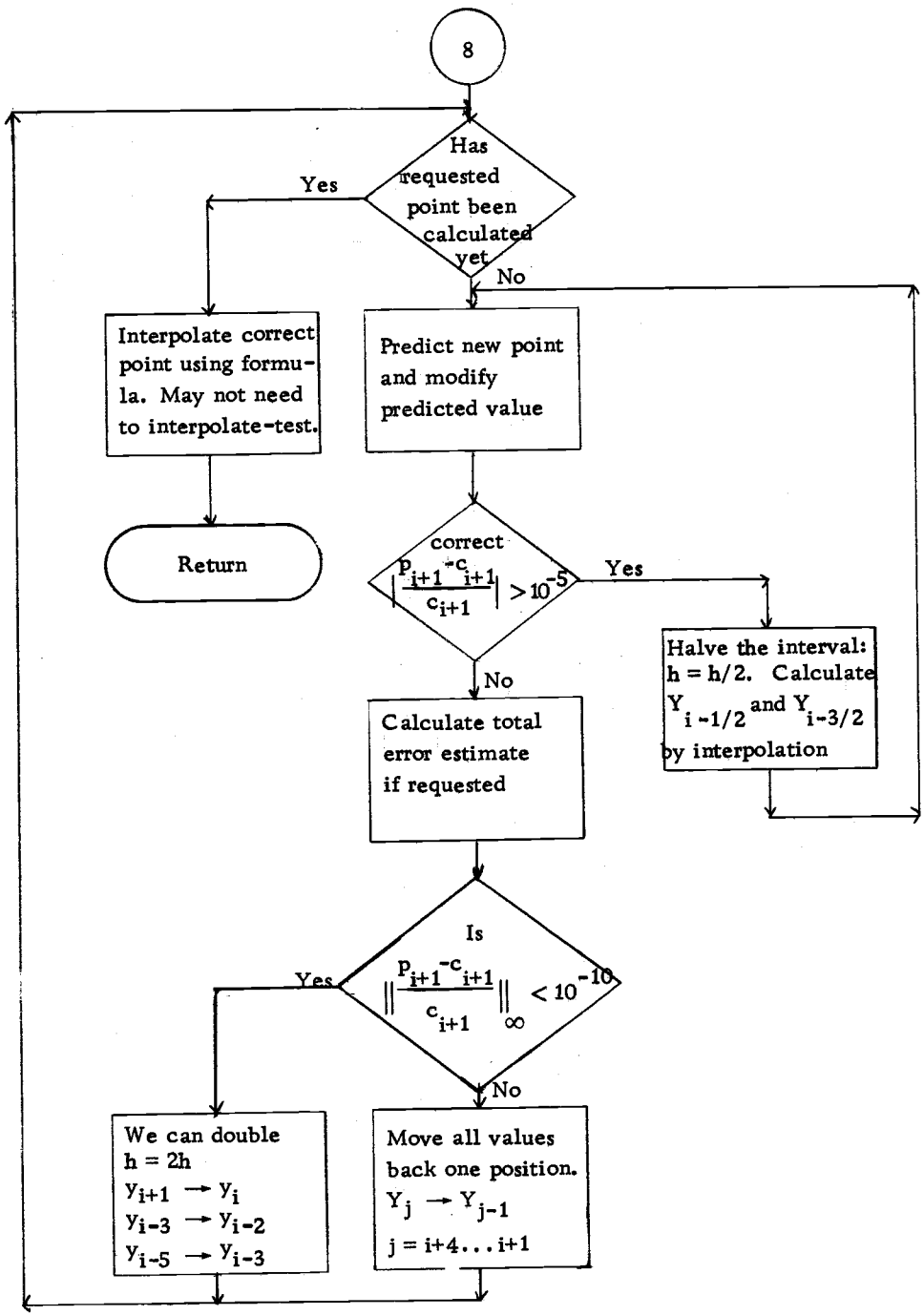

5
Right precedence of operator: Left precedence of top operator
≤
2
>
Place operator on the operator stack
B


CODGEN
Are we processing an "IF" statement
No
Yes
Initialize pointer to first operator in pseudo-code table
Scan passed the "IF" operator in the pseudo-code table
C
2
Suppress the temporary store instruction if possible (see expansion p. 76)
D
Generate the code for the operator. $, &, ; terminate. IF causes pseudo-code pointer to be set to beginning


C
Is temp indicator equal to left operand
Yes
6
No
Is temp indicator equal to right operand
No
7
Yes
Can operator be changed
No
7
Yes
Is operator a relation
No
Yes
Is this the first relation
No
Yes
Generate temp store instruction
Interchange operands. Replace operator
6


6
Suppress
left operand
D
7
Generate
instruction to store
into temp
location
D


DIFSOL
Is this first call for system of equations
No
8
Yes
Calculate Y_1, $Y_{1/2}$ and $\bar{Y}_1$ using EULER with h and h/2
Is $\left\| \frac{Y_1 - \bar{Y}_1}{Y_1} \right\|_\infty < 10^{-5}$
No
Set h = h/2
Yes
Calculate $Y_{3/2}$, iterate these to convergence
Calculate $\bar{Y}_2$, $Y_{5/2}$, $\bar{Y}_3$ Iterate these to convergence
Calculate Y_2 and Y_3 Iterate Y_1, Y_2 and Y_3
Set $Y_i = \bar{Y}_i + \frac{\bar{Y}_i - Y_i}{15}$ i = 1, 2, 3
8


8
Has requested point been calculated yet
Yes
No
Interpolate correct point using formula. May not need to interpolate-test.
Predict new point and modify predicted value
Return
correct $\left|\frac{p_{i+1}-c_{i+1}}{c_{i+1}}\right| > 10^{-5}$
Yes
Halve the interval: h = h/2. Calculate $Y_{i-1/2}$ and $Y_{i-3/2}$ by interpolation
No
Calculate total error estimate if requested
Is $\left\|\frac{p_{i+1}-c_{i+1}}{c_{i+1}}\right\|_{\infty} < 10^{-10}$
Yes
No
We can double h = 2h
$y_{i+1} \rightarrow y_i$
$y_{i-3} \rightarrow y_{i-2}$
$y_{i-5} \rightarrow y_{i-3}$
Move all values back one position.
$Y_j \rightarrow Y_{j-1}$
$j = i+4 \ldots i+1$